Improved Techniques in the Cryptanalysis of Symmetric Primitives

Antonio Florez Gutierrez

SORBONNE UNIVERSITÉ

Inria

Improved Techniques in the Cryptanalysis of Symmetric Primitives

Thèse de doctorat

Antonio Flórez Gutiérrez

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Antonio Flórez Gutiérrez

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Improved Techniques in the Cryptanalysis of Symmetric Primitives

dirigée par María Naya-Plasencia

Soutenue publiquement le 30 septembre 2022

devant le jury composé de:

María Naya-Plasencia	Inria de Paris	Directrice
Joan Daemen	Radboud University	Rapporteur
Henri Gilbert	ANSSI	Rapporteur
Charles Bouillaguet	Sorbonne Université	Examinateur
Anne Canteaut	Inria de Paris	Examinatrice
Gregor Leander	Ruhr University Bochum	Examinateur
Kaisa Nyberg	Aalto Univeristy	Examinatrice
François-Xavier Standaert	UCLouvain	Examinateur

Cover illustration: *Apatura iris* male, by F. Nemos, in: *Europas bekannteste Schmetterlinge. Beschreibung der wichtigsten Arten und Anleitung zur Kenntnis und zum Sammeln der Schmetterlinge und Raupen*, Oestergaard Verlag, Berlin, ca. 1895

Caminante, son tus huellas
el camino y nada más;
caminante, no hay camino,
se hace camino al andar.
Al andar se hace camino,
y al volver la vista atrás
se ve la senda que nunca
se ha de volver a pisar.
Caminante, no hay camino,
sino estelas en la mar.

Wayfarer, only your footprints
are the path, and nothing more;
wayfarer, there is no path,
you make the path as you walk.
As you walk you make the path,
and as you turn to glance behind
you see the trail that you never
shall return to tread again.
Wayfarer, there is no path*,*
only wake trails *in the sea.*

Antonio Machado, *Campos de Castilla (revised edition)*, 1917

Acknowledgements

Contrary to what might first appear, the poem which opens this manuscript was not chosen to make a terrible joke, but to reflect on the last three years. It is a poem which eloquently puts forward the message that life doesn't come with clear assembly instructions, and that it requires a fair amount of improvisation. For a variety of reasons, it would have been impossible to picture myself where I am now only five years ago. Indeed, it is difficult not to make any mention of recent global events, which may have well given new meaning to the word *grateful*. It used to be a somewhat empty word reserved for this kind of write-up, but the last few years have forcefully shown us that we should take nothing for granted. It is my hope that this will convince those mentioned in the following paragraphs that my feeling of indebtedness is sincere.

First and foremost, I would like to show my utmost gratitude to María Naya-Plasencia, my thesis advisor. You opened my eyes to the fascinating world of symmetric cryptography almost four years ago, which is a gift I now consider priceless. I wish I could have also borrowed your boundless optimism, but it would appear it is truly one-of-a-kind.

Similarly, I wish to thank all the other members of the jury: Charles Bouillaguet, Anne Canteaut, Joan Daemen, Henri Gilbert, Gregor Leander, Kaisa Nyberg and François-Xavier Standaert. I would like to give special thanks to Joan Daemen and Henri Gilbert for pushing their way through this manuscript, and for their thorough comments and suggestions. In addition, I am grateful to Brice Minaud and Damien Vergnaud for performing the two comités de suivi.

I would also like to express my thanks to the almost innumerable teachers and professors, both in Spain and in France, who have over the years led me along this path, and without whom this manuscript would never have come to fruition. I firmly believe that me and my classmates' futures could not have been placed in better hands.

I'd also like to mention all the researchers with whom I've had the pleasure of collaborating on or even discussing research over the past three years. In particular, I would like to show my gratitude to Kaisa Nyberg for accepting my help writing the chapter on linear cryptanalysis for the upcoming *Encyclopedia of Cryptology*, as well as Sasaki Yu and Todo Yosuke for accepting to take me in as a postdoctoral researcher at NTT.

Words cannot even begin to describe the positivity of the environment at the Équipe COSMIQ in the Paris Inria center, to whose members old and new I would also like to extend my thanks to. Alex, André (le Vieux), André (le Jeune), Andrea, Anne, Anthony, Antoine, Augustin, Aurélie, Aurelien, Charles, Christelle, Christina, Clara, Clémence, Daniel, Fanny, Ferdinand, Gaëtan, Ivan, Jean-Pierre, Johanna, Jules, Kévin, Léo, Löic, Lucien, María, Matthieu, Nicholas, Nicolas, (le Petit) Nicolas, Mathias, Pascale, Paul, Pierre, Quentin, Rémi, Ritam, Rocco, Sébastien, Shibam, Shizhu, Simona, Sohaïb, Thomas, Valentin, Virgile, Virginie, Vivien, Xavier, Yann: you were the true reason I kept coming back to the lab. I hope that this is but the beginning of our collaborations.

I wish to reserve a short paragraph to the members of the so-called "1pm squad". Our conversations were often the highlight of the day, and I wish that circumstances would have allowed us to spend more time together. You will always be welcome to visit, no matter where I am.

Also of special mention are all the Spanish students who have accompanied me over these four years of extreme survival in the wastelands of this lawless city (*a.k.a.* Paris). Antonio, Diego, Enrique, Laura, María, Pablo, and Ruth: our friendships extend years into the past, and I hope that they will also extend years into the future. Emmanuel, Quentin and Théophile: you are not technically Spanish, but you also fit comfortably in this list.

A papá, a mamá, y a mis abuelos: muchas gracias por absolutamente todo.

Contents

Présentation des Travaux

Chapitres 1 à 3

La première partie de ce manuscrit présente une introduction aux sujets principaux nécessaires pour la compréhension de la deuxième partie. Dans le chapitre 1, nous discutons de façon très rapide quelques aspects fondamentaux de la théorie de la Cryptologie Symétrique : l'histoire de la Cryptologie, les schémas de chiffrement par bloc et en particulier le DES, et la cryptanalyse différentielle. Dans le chapitre 2, nous proposons une étude plus précise de la cryptanalyse linéaire, qui est très importante pour mes sujets de recherche. Nous avons essayé de représenter la théorie classique, mais aussi des développements plus récents, pour donner une vision globale du domaine. Finalement, le chapitre 3 est consacré à la transformée de Walsh, en tant qu'opération sur les vecteurs complexes dans $\mathbb{C}\mathbb{F}_2^n$ et en tant que transformation sur les fonctions booléennes. Dans ce chapitre, nous faisons aussi une analyse détaillée de l'algorithme de la transformée de Walsh rapide (FWHT).

Chapitre 4

Ce chapitre présente une partie des résultats que j'ai introduit dans le papier [Fló22] en détail. Nous nous demandons comment nous pouvons élaguer (*prune* en anglais) l'algorithme de la transformée de Walsh rapide quand les entrées non-nulles et les sorties que nous voulons calculer font partie des sous-variétés affines de $\mathbb{F}_2^n$. Plus précisément, nous nous interrogeons sur le problème suivant :

Définition (Élagage affine de la transformée de Walsh). Soit $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ un vecteur complexe de longueur 2^n. Nous avons des listes $L, M \subseteq \mathbb{F}_2^n$, des sous-espaces vectoriels $X, U \subseteq \mathbb{F}_2^n$ et des vecteurs $x_0, u_0 \in \mathbb{F}_2^n$ tels que $L \subseteq x_0 + X, M \subseteq u_0 + U$, et $f(x) = 0$ pour tout $x \notin L$. Nous nous demandons comment calculer $\widehat{f}(u)$ pour tout élément $u \in M$ avec le moins d'opérations possible.

Nous proposons trois essais pour résoudre ce problème avec différents degrés de succès. Nous avons proposé le premier dans [FN20], et cette approche est utilisable quand les sous-variétés sont générées par vecteurs dans la base canonique. Le deuxième n'a pas été publié et repose sur l'idée de construire des algorithmes rapides pour la transformée de Walsh paramétrés par une famille de bases de $\mathbb{F}_2^n$. Finalement, nous décrivons la solution générale que j'ai introduit dans [Fló22]. L'idée de cet algorithme est de *réduire* la transformée objectif à une transformée de taille plus petite. À la fin, nous obtenons le résultat suivant :

Theorème. Dans les conditions de la définition précédente, on considère la valeur $t = \dim\left(X/(X \cap U^{\perp})\right) = \dim\left(U/(U \cap X^{\perp})\right)$. Il existe un algorithme qui peut calculer $\widehat{f}(u)$ pour tout $u \in M$ avec

$$|L| + t2^t + |M| \text{ opérations.}$$

Nous montrons aussi que l'algorithme proposé est presque optimal.

Chapitre 5

Le chapitre 5 présente les améliorations des algorithmes pour la cryptanalyse linéaire qui ont été proposées dans [FN20] et [Fló22]. Ces nouveaux algorithmes sont basés sur les travaux de Collard, Standaert et Quisquater [CSQ07b], qui proposent une méthode pour les attaques de récupération de clef (*key recovery attack* en anglais).

L'Algorithme 2 de Matsui [Mat93] est un des piliers de la cryptanalyse linéaire. Nous considérons une approximation linéaire de la forme $\langle \alpha, x \rangle + \langle \beta, \widehat{y} \rangle$, où x est le texte clair et $\widehat{y}$ est l'état quelques tours avant l'obtention du chiffré y. L'idée est que nous pouvons deviner une petite partie de la dernière clef

Table 1: Résumé des attaques sur des schémas de chiffrement par bloc décrits dans cette thèse. n : taille de bloc. κ : taille de la clef. r : nombre de tours attaqués.

					Complexité			
Schéma	n	κ	r	**Type**	**Données**	**Temps**	**Mémoire**	**Source**
PRESENT	64	80	26	Linéaire	$2^{61.1}$ KP	$2^{68.2}$	$2^{44.0}$	[FN20]
			26	Linéaire	$2^{60.8}$ KP	$2^{71.8}$	$2^{44.0}$	[FN20]
			27	Linéaire	$2^{63.4}$ DKP	$2^{72.0}$	$2^{44.0}$	[FN20]
			28	Linéaire	$2^{64.0}$ DKP	$2^{77.4}$	$2^{51.0}$	[FN20]
		128	28	Linéaire	$2^{64.0}$ DKP	2^{122}	$2^{84.6}$	[FN20]
			29	Linéaire	$2^{64.0}$ DKP	$2^{124.06}$	$2^{99.2}$	[Fló22]
DES	64	56	16	Linéaire	$2^{41.50}$ KP	$2^{42.13}$	$2^{38.75}$	[Fló22]
NOEKEON	128	128	12	Linéaire	2^{119} KP	$2^{120.85}$	2^{112}	[BCF+21]
Serpent	128	256	12	Différentielle-linéaire	$2^{127.92}$ CP	$2^{233.55}$	$2^{127.92}$	[BCD+22]
			12	Différentielle-linéaire	$2^{125.74}$ CP	$2^{236.91}$	$2^{125.74}$	[BCD+22]
			12	Différentielle-linéaire	$2^{118.40}$ CP	$2^{242.93}$	$2^{118.40}$	[BCD+22]

de tour pour calculer la valeur de l'approximation. Si nous utilisons une approximation biaisée et un nombre suffisant de pairs clair-chiffré, cela permet de séparer la bonne valeur de cette partie de la clef.

Le coût naïf de ce type d'attaque est $\mathcal{O}\left(N2^l\right)$, où N est le nombre de paires clair-chiffré que nous utilisons et l est le nombre de bits de clef que nous devons deviner. Dans [Mat94a], Matsui a aussi proposé une version alternative avec une complexité de $\mathcal{O}\left(N\right) + \mathcal{O}\left(2^{2l}\right)$. La version basée sur la transformée de Walsh rapide [CSQ07b] a une complexité de $\mathcal{O}\left(N\right) + \mathcal{O}\left(l2^l\right)$. L'attaque consiste en deux phases : au début, nous construisons une table contenant les informations relevantes relatives aux paires clair-chiffré. Dans la deuxième phase, nous appliquons la transformée de Walsh sur cette table, nous multiplions ses entrées par une table précalculée d'autovaleurs, puis nous appliquons la transformée de Walsh une deuxième fois.

Nous considérons deux familles d'améliorations sur ce type d'attaque :

- Premièrement, nous considérons des améliorations qui n'utilisent aucune modification de l'algorithme de la transformée de Walsh rapide en lui-même. Dans [FN20], nous avons proposé une version matricielle de l'attaque de Collard et al. adaptée pour un nombre arbitraire de tours de récupération de clef au début et à la fin du schéma. Nous avons généralisé cette version dans [Fló22] vers une version qui peut utiliser des vecteurs de dimension arbitraire. Nous avons aussi considéré les adaptations nécessaires de l'algorithme pour l'utiliser dans le contexte des attaques linéaires multiples, comme nous avons montré dans [FN20].
- Nous considérons aussi des améliorations basées sur l'élagage de la transformée de Walsh rapide. Le *key schedule* induit des relations qui peuvent être utilisées pour réduire la complexité des transformées de Walsh, comme nous avons montré dans[FN20]. Dans la même objectif, nous pouvons aussi utiliser les zéros dans les spectres de Walsh des fonctions de tour avec le même but.

Ces contributions sont utilisées pour proposer de nouvelles attaques sur PRESENT (jusqu'à 28 tours pour la version de 80 bits et jusqu'à 29 tours pour la version de 128 bits), le DES et NOEKEON (jusqu'à 12 tours). Ces attaques sont présentées dans le tableau 1. Dans le cas de PRESENT, les nouvelles attaques sont les premières sur 28 et 29 tours. Dans le cas du DES et de NOEKEON, nous proposons les attaques avec la meilleure complexité en données par rapport à ce qui a déjà été proposé dans la littérature.

Chapitre 6

Le chapitre 6 met en avant le travail coopératif avec Marek Broll, Federico Canale, Nicolas David, Gregor Leander, María Naya-Plasencia et Yosuke Todo qui ont été publiés dans [BCF+21] et [BCD+22]. Le but est d'améliorer la complexité des attaques de récupération de clef en utilisant les propriétés des boîtes S attentivement. Dans ce type d'attaques, il arrive souvent que nous ayons besoin d'une partie de la

sortie d'une boîte S, par exemple un seul bit. Pour déterminer cette partie de la sortie de la boîte S, nous n'avons pas toujours besoin de tous les bits d'entrée. Cela nous permet de réduire le nombre de bits de clef que nous devons deviner.

Nous nous intéressons particulièrement aux descriptions des boîte S utilisant des diagrammes ou arbres de décision binaires (*binary decision diagrams* ou BDDs). Chaque nœud demande un bit de parité de l'entrée, et nous choisissons une branche selon sa valeur. Après quelques requêtes, nous arrivons sur une feuille puis nous récupèrons la valeur de la fonction souhaitée. Nous pouvons mesurer le coût d'une attaque utilisant cette stratégie par rapport à l'attaque "naïve" à partir du quotient entre le nombre de feuilles de l'arbre (dénoté nbleaves) et le nombre d'entrées possibles 2^n.

Nous proposons des algorithmes qui trouvent des diagrammes optimaux pour une fonction booléenne, et nous discutons l'utilisation de ce type de technique pour les familles d'attaques plus connues : la cryptanalyse différentielle et la cryptanalyse linéaire. Finalement, nous décrivons les attaques différentielles-linéaires sur 12 tours de Serpent qui ont été proposées dans [BCD+22], et qui se trouvent aussi dans le tableau 1.

Chapitre 7

Dans ce dernier chapitre, nous discutons les approximations linéaires de la permutation Gimli qui ont été proposées dans [FLN+20] (Best Paper Award à Asiacrypt 2020) et [FLN+21]. Ces papiers sont issus d'une collaboration avec Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher et Ferdinand Sibleyras, où nous proposons diverses cryptanalyses des primitives symétriques basées sur Gimli.

Parmi ces contributions, j'ai réalisé une étude des propriétés linéaires de Gimli en utilisant des techniques basées sur la modélisation avec MILP [MWGP11]. Le chapitre commence avec un résumé de l'utilisation de ces techniques, après lequel nous proposons l'étude de Gimli. Nous montrons l'existence d'une approximation linéaire avec une corrélation carrée estimée de $2^{367.6}$ sur la permutation complète.

Publications

[FN20] Antonio Flórez-Gutiérrez and María Naya-Plansecia. Improving key-recovery in linear attacks: Application to 28-round PRESENT. In *Advances in Cryptology - EUROCRYPT 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 221–249. Springer, 2020.

[FLN+20] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sybleyras. New results on Gimli: Full-permutation distinguishers and improved collisions. In *Advances in Cryptology - ASIACRYPT 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2020. *(Best Paper Award)*

[FLN+21] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sybleyras. Internal symmetries and linear properties: Full-permutation distinguishers and improved collisions on Gimli. *Journal of Cryptology*, 34(4):45, 2021.

[BCF+21] Marek Broll, Federico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, and María Naya-Plasencia. Generic framework for key-guessing improvements. In *Advances in Cryptology - ASIACRYPT 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 453–483. Springer, 2021.

[BCD+22] Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. New attacks from old distinguishers: improved attacks on serpent. In *Topics in Cryptology - CT-RSA 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 484–510. Springer, 2022.

[BBC+22] Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. Improved Differential-Linear Attacks with Applications to ARX Ciphers. *Journal of Cryptology* (accepted), 2022.

[Fló22] Antonio Flórez-Gutiérrez. Optimising Linear Key Recovery Attacks with Affine Walsh Transform Pruning. *Advances in Cryptology - ASIACRYPT 2022*.

[NF22] Kaisa Nyberg and Antonio Flórez-Gutiérrez. Linear Cryptanalysis. In *Encyclopedia of Cryptology*.

Mathematical Notation

- $|A|$ denotes the cardinal or number of elements of a set A.
- Given a matrix A over any field, we will denote the element in the i-th row and the j-th column a_{ij}.
- Given two matrices A, B over any field of sizes $m \times n$ and $p \times q$, the Kronecker product, denoted $A \otimes B$, is the $mp \times nq$ block matrix:

$$A \otimes B = \left(\begin{array}{c|c|c} a_{11}B & \cdots & a_{1m}B \\ \hline \vdots & & \vdots \\ \hline a_{n1}B & \cdots & a_{nm}B \end{array}\right)$$

- For arrays of arbitrary dimension d, we will use the notation $X[i_1, i_2, \ldots, i_d]$ to access individual entries. Sometimes it will also be used for vectors and matrices to avoid confusion, especially when there are other subscripts.
- $\mathrm{GL}(U, V)$ denotes the group of all linear maps $L : U \longrightarrow V$ between the vector space U and the vector space V.
- Let $f : A \times B \longrightarrow C$ be any map defined on a product of sets $A \times B$. Given $y \in B$, we will denote by $f(\cdot, y)$ the trace map, that is:

$$\begin{array}{rccc} f(\cdot, y): & A & \longrightarrow & C \\ & x & \longmapsto & f(x, y) \end{array}$$

Binary vectors

- The field with two elements is denoted $\mathbb{F}_2 = \{0, 1\}$. We denote the addition in this field by $x + y$ and the product by $x \cdot y$ or xy. For additions of elements of $\mathbb{F}_2$ as integers (that is, considering $1 + 1 = 2$) we will use the symbol $x \boxplus y$.
- The binary vector space of dimension n is denoted $\mathbb{F}_2^n$. Given $x \in \mathbb{F}_2^n$ and $i \in \{n-1, \ldots, 0\}$, x_i denotes its i-th coordinate from the right, where x_0 is the least significant and x_{n-1} is the most significant bit. Bitwise addition or XOR in $\mathbb{F}_2^n$ is denoted by $x + y$, although in some cases such as cipher specifications we use $x \oplus y$.
- If $x, y \in \mathbb{F}_2^n$ are two binary vectors, then $\langle x, y \rangle = \sum_{i=0}^{n-1} x_i \cdot y_i$ denotes their inner or dot product. If $\langle x, y \rangle = 0$ we say x and y are orthogonal, which we denote $x \perp y$.
- Given a binary vector $x \in \mathbb{F}_2^n$, $\mathrm{wt}(x) = \boxplus_{i=0}^{n-1} x_i$ denotes its Hamming weight or number of non-zero coordinates.
- Given $x, \chi \in \mathbb{F}_2^n$, $x|_\chi \in \mathbb{F}_2^{\mathrm{wt}(\chi)}$ will denote the vector of length $\mathrm{wt}(\chi)$ whose components are the coordinates of x corresponding to non-zero entries of χ.
- Given some binary vector of arbitrary length x, we will denote said length by $|x|$.
- Given two binary vectors $x = (x_{n-1}, \ldots, x_0) \in \mathbb{F}_2^n$ and $y = (y_{m-1}, \ldots, y_0) \in \mathbb{F}_2^m$, we will denote by $(x|y) = (x_{n-1}, \ldots, x_0, y_{m-1}, \ldots, y_0)$ the vector in $\mathbb{F}_2^n \times \mathbb{F}_2^m = \mathbb{F}_2^{n+m}$ which is obtained by concatenating the coordinates of x and y.

Table 2: Definitions, notations and properties of some probability distributions.

Bernoulli distribution Be(p)

A discrete distribution which takes value 1 with probability p and 0 otherwise.

Probability function: $Pr(X = 1) = p, \quad Pr(X = 0) = 1 - p$

Expected value: p **Variance:** $p(p-1)$

Binomial distribution $\mathcal{B}(p, n)$

The sum of n independent Bernoulli random variables Be(p).

Probability function: $Pr(X = k) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}, \quad k = 0, \ldots, n$

Expected value: np **Variance:** $np(p-1)$

For a large value of n and a value of p not too close to 0 or 1, this distribution can be approximated by the normal distribution of the same expected value and variance.

Hypergeometric distribution $\mathcal{HG}(K, N, n)$

The distribution of the number of successes among $n \leq N$ draws without replacement of a population of size N containing $K \leq N$ marked elements.

Probability function: $Pr(X = k) = \dfrac{\left(\begin{array}{c} N \\ k \end{array} \right) \left(\begin{array}{c} N-K \\ n-k \end{array} \right)}{\left(\begin{array}{c} n \\ k \end{array} \right)}, \quad k = 0, \ldots, n$

Expected value: $n\frac{K}{N}$ **Variance:** $n\frac{K}{N}\frac{N-K}{N}\frac{N-n}{N-1}$

This distribution can be approximated by the normal distribution of the same expected value and variance, although the resulting error depends on the parameters.

Normal distribution $\mathcal{N}(\mu, \sigma^2)$

Probability density function: $\phi_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad \phi = \phi_{0,1}$

Cumulative distribution function: $\Phi_{\mu,\sigma^2}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad \Phi = \Phi_{0,1}$

Expected value: μ **Variance:** σ^2

Folded normal distribution $\mathcal{FN}(\mu, \sigma^2)$

The distribution of the absolute value of a normal random variable of the same parameters.

Probability density function: $\frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} + e^{-\frac{1}{2}\left(\frac{x+\mu}{\sigma}\right)^2} \right)$

Non-central chi-square distribution $\chi^2_m(\lambda)$, $\chi^2_m = \chi^2_m(0)$

The distribution of the sum of m independent variables $\mathcal{N}(\mu_i, 1)$ verifying $\lambda = \sum_{i=1}^{m} \mu_i^2$. The distribution is said to have m degrees of freedom and non-centrality parameter λ.

Cumulative distribution function: $\Psi_{m,\lambda}(x), \quad \Psi_m = \Psi_{m,0}$

Expected value: $m + \lambda$ **Variance:** $2(m + 2\lambda)$

For a large enough value of m, this distribution can be approximated by the normal distribution of the same expected value and variance.

Part I

PRELIMINARIES

Chapter 1

Introduction to Symmetric Cryptography and Cryptanalysis

This first chapter acts as a brief overview of some fundamental concepts in the field of symmetric cryptology, with an intentional focus on those of particular interest to the contributions which constitute the second half of this thesis. The topic of linear cryptanalysis, which is integral to several chapters, is covered in-depth in Chapter 2.

Cryptography is the practice of protecting communications against a malicious adversary (as opposed to, for example, natural noise). This includes preserving the secrecy of communications against eavesdroppers, but also covers methods which provide other types of protection, such as techniques which impede an adversary from impersonating a friendly party. *Cryptanalysis* is the name given to all the techniques which aim to locate weaknesses in cryptographic schemes. The name *cryptology* is often used to denote the science which encompasses both fields.

Contents

Section 1.1 provides some basic historical context on the development of cryptology and cryptanalysis. Section 1.2 is an overview of the common constructions which are used in symmetric cryptography, with a specific focus on block ciphers. It also contains the specifications for the DES [DES77] as a concrete example. Finally, Section 1.3 outlines the theory of the cryptanalysis of block ciphers, and explains the principles of differential cryptanalysis.

1.1 History of Cryptology

The human desire to protect secrets is probably as old as their ability to communicate. Although whispering may have provided an acceptable solution to our far ancestors, the invention of the written word led to the following problem: how to secure communications when they take place through letters or other written messages which may be intercepted. Ancient history and legend provide several examples

of techniques used to transform or *encrypt* messages or *plaintexts* into seemingly nonsensical *ciphertexts*, which can nonetheless be turned back into the original message if a secret *decryption* trick is known. Most of these methods rely on either the *transposition* or rearrangement of the message's symbols or the *substitution* of the symbols by other symbols in the same or a different alphabet. However, these early cryptographers would soon find their methods compromised by the first cryptanalysts. For example, simple substitution ciphers can be broken by studying the relative frequencies of each of the symbols, as they should mimic the relative distributions of the letters of the alphabet. The history of cryptology is the history of the back-and-forth between cryptography and cryptanalysis.

One of the fundamental principles of cryptography was stated by the Dutch scholar Auguste Kerchkoffs [Ker83] in the late 19th century, and has come to be called Kerchkoffs's principle or desideratum. It states that the security of a cryptographic scheme must never rely on the obfuscation of the scheme itself, and must instead depend on a randomly generated *key*. Such an encryption scheme has two inputs: in addition to the plaintext, it takes a key, which is a random sequence of characters (for example, a string of bits); and a single output, which is the ciphertext. Without the key, it is impossible to recover the plaintext from the ciphertext. This ensures that the scheme will still be secure even if an eavesdropper has full knowledge of the way it was designed. In general, a cryptographer must never underestimate the capabilities of a potential adversary. The use of keys also has practical advantages: for example, keys can be changed periodically so that, should one of them fall in the hands of the adversary, they would still only be able to access the portion of the correspondence in which that specific key was used.

The Second World War constitutes a watershed moment in the history of cryptology. The Germans conceived the Enigma and Lorentz machines to be used in their field and high-level communications, respectively. Both machines performed an elaborate substitution cipher by means of passing electrical signals through moving rotors and a telephone-like switchboard. These systems were broken thanks to persistent efforts by Polish and British cryptanalysts, whose achievements would sadly remain classified and largely unrecognised by the public for decades after the end of the War. These cryptanalysis activities were only possible thanks to the development of the first electromechanical and fully electronic computers. From this point onwards, computers would become essential tools to both cryptography and cryptanalysis.

Another field which would prove to have a very close relationship with cryptology was mathematics. The theoretical foundations of cryptography would be laid down thanks to the development of information theory. In particular, Shannon [Sha48] showed that it is possible to construct an *unconditionally secure* cipher, in the sense that any adversary is provably unable to deduce anything about the plaintext from the ciphertext without any prior information about the key. This cipher is known as the one-time pad, equivalent versions of which had already been described by Miller and Vernam [Mil82, Ver19]. The one-time pad encrypts a plaintext by substituting each symbol with another in the alphabet according to a key that's as long as the message itself. For example, if both the message and the key are bit strings, the ciphertext is obtained by xoring the key to the plaintext. Although it has very good theoretical properties, it has several practical problems. In particular, it requires the secure exchange of keys which have the same length as the message, and information about the plaintext can leak if the key is not generated in an adequate manner [Ben12].

Modern cryptography understands security in terms of computational cost: most if not all current cryptosystems can be broken by following some kind of generic attack method. For example, if the key consists of a string of 128 bits, an adversary may just try to decrypt a message using all the possible keys until they obtain a plaintext which makes sense. This would require the attacker to perform 2^{128} test decryptions, which is generally considered to be out of reach to the computers of today and the near future. Of course, the possibility still remains that some unknown shortcut exists which would permit an attack to be performed at a much smaller cost. It is the role of cryptographers to make their designs known to the community, and the role of cryptanalysts to examine them to try and locate such vulnerabilities. Confidence in a construction should be only as deep as the community's scrutiny of it.

There are three main security goals a cryptographic system may wish to accomplish:

- **Confidentiality:** The interlocutors must be able to exchange encrypted messages which, even if intercepted by a third party, would reveal none of their contents.
- **Authenticity:** The interlocutors must be able to confirm each other's identities when exchanging messages so that it is impossible for a third party to impersonate them.
- **Integrity:** It must be impossible for a third party to alter the content, replace or forge messages between the interlocutors without being noticed.

For example, the historical encryption methods discussed in the paragraphs above mainly aimed to achieve confidentiality. They also provide some degree of authenticity and integrity, as an adversary

would be unable to encrypt their own fake plaintexts and produce decryptable ciphertexts. However, an attacker may obtain a copy of a ciphertext and then send it at a later time pretending to be a legitimate interlocutor, or even use the old ciphertext to substitute a new message (for example, swapping a message saying "Attack now" for a copy of an earlier message saying "Stand by" may have massive real-world consequences). This kind of exploit highlights the fact that well-designed cryptographic constructions may still be vulnerable if they are used as part of badly-designed protocols. There are also situations in which confidentiality may not be desired, but other aspects may be of interest: for example, someone may wish to prove their authorship of a publicly available document by means of an attached *digital signature*.

Another consequential development was the introduction of *public-key* or *asymmetric* cryptography in the late 70s. Traditional cryptography is *private-key* or *symmetric*, in the sense that all (pairs of) interlocutors share the exact same key and have the same encryption and decryption capabilities. The fundamental concept of asymmetric cryptography was outlined by Diffie and Hellman [DH76], and the first public-key cryptosystem was introduced by Rivest, Shamir and Adleman [RSA78]. Asymmetric cryptography uses pairs of associated keys which are linked by some mathematical problem which is considered difficult to solve. For example, RSA uses a private key which consists of two large prime numbers, and both primes are multiplied to obtain the public key. Reconstructing the private key from the public key is achieved through factorization, which is considered to be a hard problem. The public key, as its name implies, can be distributed to the public, while the private key is kept by the person who generated it. In the case of encryption, anyone can use the public key to send messages, but the private key, which is held only by the intended recipient, is required in order to decrypt them. In the case of digital signatures, only the private key can be used to generate signatures, but anyone can use the public key to check their validity.

There are several important differences between public-key and symmetric cryptography beyond their fundamental design purpose which heavily influence their deployment in the real world. The security of asymmetric constructions tends to rely on mathematical problems which are believed to be of high computational complexity, such as factorisation or the discrete logarithm. On the other hand, symmetric constructions often lean more heavily towards the lack of known attacks as a security argument: if no vulnerabilities have been found after years of close examination by cryptanalysts, then a construction may be considered trustworthy. However, this is a blurry line: asymmetric cryptology also uses cryptanalysis efforts as a security argument, and there are many symmetric constructions with security proofs. Another significant difference is the computational cost: public-key cryptography most often requires more computational resources to implement. Many applications thus use public-key cryptography as a key establishment tool, so that communication can continue using private-key cryptography.

Perhaps the most fundamental shift to be experienced by cryptology in recent decades has been the gradual move towards public research and widespread use. Until the mid-20th century, cryptography was a field very closely associated with the affairs of the military and the state, and most research was carried out behind closed doors. The development of computers and later computer networks and the Internet generated a lot of interest in cryptography from the private sector at first and from public research institutions shortly after. This increased interest eventually led to the formation of a community focusing on the development (and sometimes rediscovery) of ideas and techniques in cryptography and cryptanalysis with the sole aim of benefiting the general public. This ever-expanding library of public knowledge would eventually prove essential to the current age of computer-based communication.

1.2 Symmetric Cryptography

This section presents the basics of block cipher construction and use, and as an example, we describe the specification of the DES [DES77]. We also discuss other common constructions which fall under the umbrella of symmetric cryptology, such as stream ciphers and hash functions.

1.2.1 Block Ciphers

Block ciphers are one of the most commonly-used constructions in symmetric cryptography. A block cipher encrypts plaintexts or *blocks* of a fixed number of bits, usually 64, 128 or 256, using a secret key which usually also has a fixed length.

Definition 1.1 (Block cipher). *A* block cipher *E of block length n and key size κ is a map $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ for which $E_K = E(\,\cdot\,, K)$ is a bijective map (often called* permutation *in the cryptology literature) for any* key *$K \in \mathbb{F}_2^\kappa$. Given a fixed key K, this map takes a* plaintext *x and returns a* ciphertext

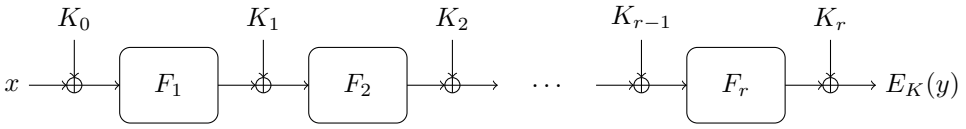


Figure 1.1: Diagram of an r-round key-alternating block cipher.

$y = E_K(x)$. *The map* E_K *is called the* encryption map. *Meanwhile, for every key* K*, there exists an inverse* decryption map *map* E_K^{-1}. *We also write* $E^{-1}(y, K) = E_K^{-1}(y)$ *as a slight abuse of notation.*

Of course, not every map E which satisfies the previous definition would make for a useful block cipher, as there are several performance and security constraints which must be satisfied. In terms of performance, when the key K is known, the encryption and the decryption operations should be possible to perform at a very low computational cost. This ensures that large quantities of data can be encrypted and decrypted in a short time in a multitude of application scenarios. On the other hand, we also wish the block cipher to be secure. For example, it should be computationally difficult to obtain any information about the key K by examining pairs of plaintexts and ciphertexts or by querying a black box which computes E_K. Ideally, we wish the permutation E_K to be virtually indistinguishable from a randomly sampled permutation of $\mathbb{F}_2^n$, a notion which will be explored in more detail in Subsection 1.3.1.

1.2.1.1 Block Cipher Constructions

There are several common constructions which are used in many block cipher designs. Most block ciphers are *iterative* or *product ciphers*, in the sense that they consist of the consecutive application of simpler *rounds*. Each round applies some (possibly key-dependent) transformation which is easy to compute, the aim being that a secure block cipher is obtained after enough rounds have been applied. The key material which is used in each round is called the *round subkey*. The round subkeys are generated from the master key K by means of a *key schedule*. Each of the intermediate results or registers on which the round transformations are applied is often called the *state*.

Many iterative block ciphers can be described as *key-alternating* [DR02], which means they alternate between bitwise key additions and a round transformation which is independent of the key:

Definition 1.2 (Key-alternating block cipher). $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ *is an* iterative key-alternating block cipher *if it can be written as a composition of the following form:*

$$E_K = \mathrm{ARK}_{K_r} \circ F_r \circ \mathrm{ARK}_{K_{r-1}} \circ \cdots \circ \mathrm{ARK}_{K_2} \circ F_2 \circ \mathrm{ARK}_{K_1} \circ F_1 \circ \mathrm{ARK}_{K_0}, \tag{1.1}$$

where $F_i : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ *are the (bijective) round functions and* $\mathrm{ARK}_{K_i} : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ *(*ARK *standing for Add Round Key) are the round key additions, in other words,* $\mathrm{ARK}_{K_i}(x) = x \oplus K_i$. *The map* $\mathrm{ARK}_{K_i} \circ F_i$ *is called the* i*-th round. The round keys* $K_0, \ldots, K_r$ *are determined using the* key schedule $\mathrm{KS} : \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^{(r+1)n}$.

It is clear that the decryption map for a key-alternating block cipher is as follows:

$$E_K^{-1} = \mathrm{ARK}_{K_0} \circ F_1^{-1} \circ \mathrm{ARK}_{K_1} \circ F_2^{-1} \circ \mathrm{ARK}_{K_2} \circ \cdots \circ \mathrm{ARK}_{K_{r-1}} \circ F_r^{-1} \circ \mathrm{ARK}_{K_r}. \tag{1.2}$$

Please note that although a block cipher may fit Definition 1.2, the specification may differ from this description. For example, round and subkey numbering may be different, or the key addition may be considered as the first step of the round instead of the last. In other cases, the key-alternating structure may be somewhat hidden (for example, when the round subkey is only xored to part of the state). Whenever a specific block cipher is discussed in this text, we will use the round numbering and naming conventions used in the original specification.

In more specific terms, there are several typical iterative constructions which appear regularly in proposed block ciphers and which have received a lot of analysis:

- **Feistel networks:** First used by Feistel and Coppersmith while designing ciphers at IBM in the early 70s, this construction would become quite popular, and was the basis of the Data Encryption Standard or DES [DES77]. Other examples include GOST [GOS89], Blowfish [Sch93], Twofish [SKW+98], Camellia [AIK+00], CLEFIA [SSA+07], Simon [BSS+13] and Speck [BSS+13]. The state of a Feistel cipher consists of two parts of equal length, $X_i = (L_i, R_i)$. Every round, the transformation $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus f(R_i, K_i))$ is applied, where f can be any (not necessarily

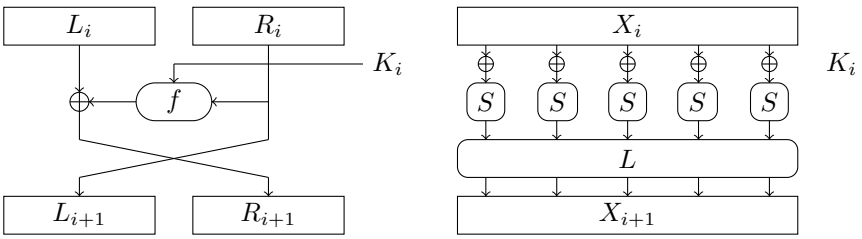


Figure 1.2: The round functions of a Feistel network (left) and an SPN (right).

bijective) function from $\mathbb{F}_2^{n/2} \times \mathbb{F}_2^k$ to $\mathbb{F}_2^{n/2}$. The security properties of this construction have been largely studied in works such as that of Luby and Rackoff [LR88], and several generalisations have been proposed, such as the Lay-Massey construction [LM90] and generalised Feistel networks.

- **Substitution permutation networks (SPNs):** This is also a construction with a long history (it was already described by Feistel in [Fei73]). The round function of an SPN consists of two operations: the parallel application of small non-linear maps called Sboxes (often called the Sbox layer or substitution layer), and a linear transformation on the whole state (often called the linear layer or permutation layer). The idea is that the Sboxes provide a highly complex but local transformation of the state (often called *confusion*), while the linear layer provides a less complex but global transformation (often called *diffusion*). SPN block ciphers often incorporate the round key by xoring it to the state, thus making them key-alternating block ciphers. The term SPN is often reserved for ciphers in which the linear layer is a permutation of the bits or the words of the state, such as with PRESENT [BKL+07], RECTANGLE [ZBL+14] and GIFT [BPP+17]. There are other ciphers which use linear layers which are not bit permutations but which are still referred to as SPN in the literature, such as the Advanced Encryption Standard or AES [AES01], Serpent [BAK98] and Noekeon [DPVAR00]. A common approach to the construction of linear layers [DR02] is to divide them into a mixing layer and a transposition. For example, the 128-bit AES state consists of 16 8-bit words which are seen as elements of $\mathbb{F}_{2^8}$ and arranged in a 4×4 grid. The Sbox layer SubBytes applies a nonlinear map to each of the 8-bit registers. The mixing layer MixColumns applies a linear transformation on each column of the grid which is given by a matrix in $\mathrm{GL}(\mathbb{F}_{2^8}^4, \mathbb{F}_{2^8}^4)$. The transposition layer ShiftRows applies a permutation on the words of each row of the grid.

- **Add-rotate-XOR (ARX) constructions:** ARX constructions divide the state into several registers of the same number of bits l and apply three basic operations on them: addition modulo 2^l, circular bit rotations and bitwise XOR. The aim is to obtain primitives with very good software performance, as all these operations are included in most processors' instruction sets. Well-known examples include the stream ciphers Salsa20 and ChaCha [Ber08] and the block cipher family SPECK [BSS+13].

These constructions can be used in primitives other than block ciphers and are not mutually exclusive, but most block ciphers adhere to at least one of them. Another important feature of cipher design is the key schedule. Badly-designed key schedules can lead to several kinds of attacks, such as meet-in-the-middle [DH77] and slide [GT78, BW99] attacks.

1.2.1.2 Block Cipher Modes of Operation

On its own, a block cipher E can only encrypt plaintexts which have a fixed length which is equal to the block size n, usually 64, 128 or 256 bits. For messages which do not fit this fixed length, it is necessary to define a mode of operation which describes how longer inputs will be handled. The message has to be divided into blocks of n bits, and in some cases we may need to define a padding scheme which extends the message length to a multiple of n. We must also choose how to feed these n-bit blocks through the block cipher. The NIST recognises five modes of operation which provide message confidentiality [NIS01]:

- **Electronic codebook (ECB):** In ECB mode (Figure 1.3), each block is encrypted independently with E. Although it has some advantages such as the ability to individually decrypt any segment of the message, it provides weak security. For example, since identical plaintext blocks result in identical ciphertext blocks, an adversary might be able to identify repeating blocks in the message.

- **Cipher block chaining (CBC):** In the widely-used CBC mode [EMST76] (Figure 1.4), the output of the encryption of each block is used to whiten the input of the next block. This has the result of erasing any patterns in the plaintext from the ciphertext. Any change in a plaintext block

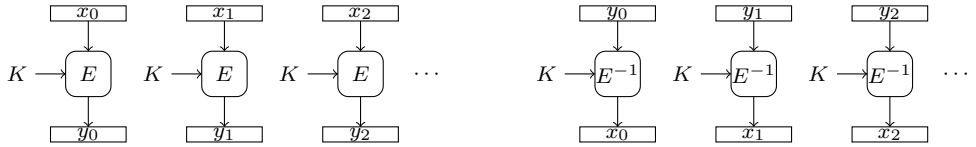


Figure 1.3: Encryption and decryption in the electronic codebook (ECB) mode.

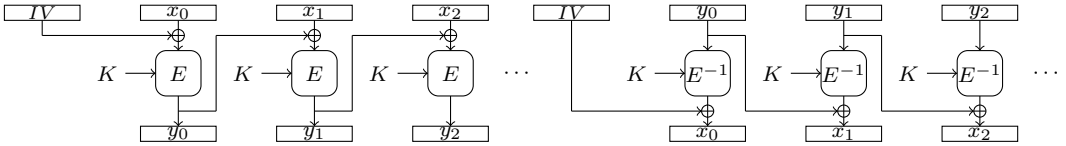


Figure 1.4: Encryption and decryption in the cipher block chaining (CBC) mode.

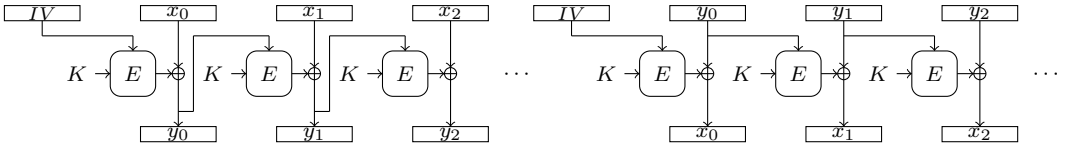


Figure 1.5: Encryption and decryption in the cipher feedback (CFB) mode.

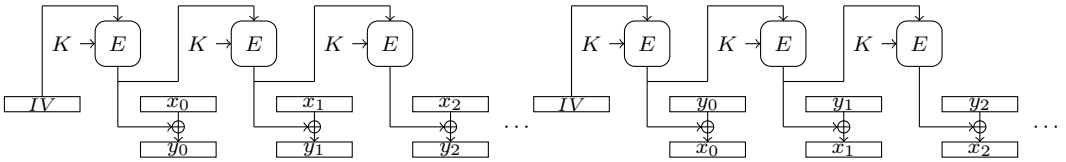


Figure 1.6: Encryption and decryption in the output feedback (OFB) mode.

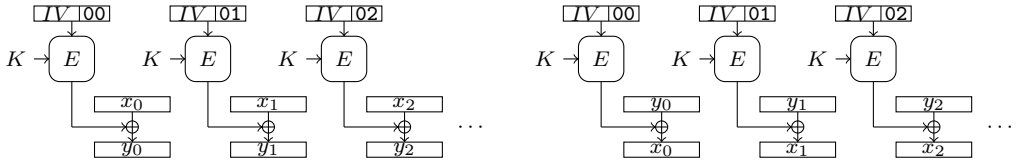


Figure 1.7: Encryption and decryption in the counter (CTR) mode.

propagates to all subsequent blocks. The first block is whitened using a random initialization vector IV. In general, the initialization vector is considered to be a nonce, that is, a non-secret string which is only used once. This also adds the advantage that the same message will produce different ciphertexts each time it is encrypted.

- **Cipher feedback (CFB):** In CFB mode (Figure 1.5), each output block of the encryption is fed through the block cipher and then xored to the next input block. This mode will also erase any patterns in the plaintext. It has some advantages over CBC mode: in particular, it is not necessary to implement the block cipher decryption, as E is used for both encryption and decryption. It also removes the need for plaintext padding, as it suffices to truncate the last output of the block cipher to the appropriate size.
- **Output feedback (OFB):** OFB mode (Figure 1.6) uses the block cipher to construct a stream cipher (see the next section). By feeding an initialisation vector through multiple iterations of E, a keystream is obtained, which can be xored to the plaintext to obtain the ciphertext, in an analogous manner to the one-time pad.
- **Counter mode (CTR):** CTR mode (Figure 1.7) also operates as a stream cipher. Each plaintext block is xored to the output of the block cipher, which is fed an input which consists of a nonce and a counter which indicates the position of the block in the plaintext. This mode has the advantage that it is possible to encrypt and decrypt individual blocks of the message independently.

In addition to these confidentiality-only modes, additional modes exist for authenticated encryption.

Along with the ciphertext, these modes produce a tag or message authentication code (MAC) which can be used to check the authenticity of the message. Some authenticated encryption schemes consist of an encryption mode with an attached tag computation, while others compute both at the same time. In recent years, attention has been given to authenticated encryption with associated data (AEAD), which permits binding some unencrypted associated data to the ciphertext, in such a way that the associated data is also authenticated.

1.2.2 Other Symmetric Constructions

In addition to block ciphers, there are other common basic primitives and constructions in symmetric cryptography, which can be used in situations in which a block cipher would be impractical (stream ciphers) or where an objective other than encryption is desired (hash functions). Since the aim of this section is just to familiarise the reader with some common cryptographic algorithms which are not block ciphers, we will not consider a rigid classification: for example, a block cipher in OFB or CTR mode can be considered a stream cipher.

1.2.2.1 Stream Ciphers

Stream ciphers are motivated by a desire to use a more practical version of the one-time pad. Instead of generating a random key which is as long as the plaintext, a shorter master or seed key is shared between the interlocutors, and some expansion function is applied to it in order to obtain a pseudorandom bit sequence, called the keystream, which can be xored to the plaintext. Stream ciphers are often simpler to implement and are frequently used when the length of the plaintext cannot be determined in advance, as happens for example in telephone calls.

Many stream cipher constructions consist of three distinct parts:

- An *initialisation function*, which takes the master key and potentially a nonce, and uses them to populate an initial internal state.
- An *update function* which transforms the internal state, and is applied repeatedly. Update functions based on linear feedback shift registers (LFSRs) and nonlinear feedback shift registers (NLFSRs) are quite popular.
- An *extraction function* which generates a segment of keystream from the state between applications of the update function.

There are other approaches to stream cipher design. ChaCha [Ber08], which is one of the most commonly-used stream ciphers, is built around an ARX-based pseudorandom permutation which is used in counter mode. The 256-bit seed key is incorporated to the input of the permutation along with the nonce and the counter.

1.2.2.2 Cryptographic Permutations

Cryptographic permutations can be used as a building block in symmetric primitives, such as the Even-Mansour construction [EM91] or the sponge and duplex constructions and their variants [BDPA07]. Cryptographic permutations are thus part of all kinds of primitives, from block and stream ciphers to hash functions and even AEAD primitives.

Informally speaking, a cryptographic permutation is a relatively easy to compute bijective map from $\mathbb{F}_2^n$ to itself which behaves like a randomly sampled permutation, in other words, a block cipher without a key. When used as a symmetric primitive, we desire that the resulting construction has the same security as if a true random permutation had been used, and that there are no attacks which make use of the specific structure of the cryptographic permutation. For example, in the case of sponges [BDPA07], this principle is called the *hermetic sponge strategy*: there should be no special properties or *structural distinguishers* specific to the permutation which could be leveraged by an attacker, so that the only possible attacks are those generic to any permutation.

Also of interest to cryptologists are pseudorandom functions, which are defined in the same way but excluding the bijection requirement. Pseudorandom functions are quite different objects from a cryptographic perspective, however, as the presence of collisions of the type $f(x) = f(y)$ can have major effects on security. It is possible to obtain a permutation from a pseudorandom function by using several different constructions, such as the Feistel network.

1.2.2.3 Cryptographic Hash Functions

Although hash functions feature no key and thus do not technically fall under neither symmetric nor asymmetric cryptography, they are often considered to be part of the former because of the similarities in both design and cryptanalysis. A hash function is an easy-to-compute map H which takes a bit string of any length as an input and outputs a hash which has a fixed length. Hash functions have a lot of uses in computer science inside and outside of cryptography. Cryptographic hash functions in particular have several applications, such as digital signatures and password management. In order to be considered cryptographically secure, a hash function must have the following properties:

1. **Preimage resistance:** Hash functions should be one-way functions. Given a target hash y, it must be difficult to find a preimage x which hashes to y, that is, $y = H(x)$.

2. **Second preimage resistance:** Given an input x, it should be difficult to find a second message x' with $x' \neq x$ so that both messages have the same hash, $H(x) = H(x')$.

3. **Collision resistance:** It should be difficult to find collisions of the hash function, that is, pairs x, x' for which $H(x) = H(x')$.

The most recent NIST standard for cryptographic hash functions consists of the SHA-3 family of hash functions [SHA15], which was the result of a selection process and comprises functions generating hashes of 224, 256, 384 and 512 bits based on the Keccak sponge construction. The same standard includes the extendable output functions (XOFs) SHAKE128 and SHAKE256, which can produce outputs of arbitrary length d.

1.2.3 The Data Encryption Standard (DES)

After the general discussion of symmetric cryptography constructions in the previous pages, we will now describe a concrete example, which also happens to be a target for cryptanalysis in Chapter 5. The Data Encryption Standard or DES is a 64-bit block cipher using a 56-bit key which was initially developed by Horst Feistel at IBM. This block cipher was presented to the US National Bureau of Standards or NBS (the organism which would eventually become the National Institute of Standards and Technology or NIST). After a few modifications, the DES standard was published in 1977 [DES77].

The DES manipulates a 64-bit state (L, R) which is divided into two 32-bit halves. The state is initially populated by rearranging the plaintext (denoted P in the specification) according to a fixed initial permutation IP. The cipher operates as a 16-round Feistel network. The round function f has a 32-bit string and a 48-bit round subkey as inputs and outputs a 32-bit string. After sixteen rounds, the left and right parts are swapped, and the inverse initial permutation is applied to obtain the ciphertext. Since the initial permutation is irrelevant to cryptanalysis, it will be ignored in the rest of this text, thus denoting $P = (L_0, R_0)$ and $C = (R_{16}, L_{16})$.

The round function $f : \mathbb{F}_2^{32} \times \mathbb{F}_2^{48} \longrightarrow \mathbb{F}_2^{32}$ consists of four steps:

1. The 32-bit input R is expanded to a 48-bit string using a fixed expansion map E. Each bit of the output string is a copy of one of the input bits, as given by the following table:

Output bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24
Input bit	0	31	30	29	28	27	28	27	26	25	24	23	24	23	22	21	20	19	20	19	18	17	16	15
Output bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input bit	16	15	14	13	12	11	12	11	10	9	8	7	8	7	6	5	4	3	4	3	2	1	0	31

2. The resulting 48-bit word is xored to the round subkey.

3. The 48-bit result is divided into eight 6-bit words. A different 6-to-4-bit Sbox $S_i : \mathbb{F}_2^6 \longrightarrow \mathbb{F}_2^4$ is applied to each one of these eight segments, S_1 being used on the leftmost part and S_8 being used on the rightmost part. The eight Sboxes are given as a set of lookup tables. They have the particularity that for any $(x_5, x_0) \in \mathbb{F}_2^2$, the map $S(x_5, \cdot, \cdot, \cdot, \cdot, x_0)$ is a bijection on $\mathbb{F}_2^4$. The result of this operation is a 32-bit string.

4. The output of f is obtained by applying a fixed bit permutation P, given by the following table, on the result of the previous step.

Algorithm 1: Data Encryption Standard Algorithm

Input: A 64-bit plaintext P. 16 48-bit round subkeys $K_i,\ i = 1, \ldots, 16$.
Output: A 64-bit ciphertext C.
$(L_0, R_0) \leftarrow IP(P)$;
for $i \leftarrow 1$ **to** 16 **do**
  $L_i \leftarrow R_{i-1}$;
  $R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, K_i)$;
end
$C \leftarrow IP^{-1}(R_{16}, L_{16})$;
return C;

Algorithm 2: DES key schedule

Input: A 64-bit master key K, including 8 parity check bits.
Output: 16 48-bit round subkeys $K_1, \ldots, K_{16}$.
$(C_0, D_0) \leftarrow PC_1(P)$;
for $i \leftarrow 1$ **to** 16 **do**
  $C_i \leftarrow LS_{p(i)}(C_{i-1})$;
  $D_i \leftarrow LS_{p(i)}(D_{i-1})$;
  $K_i \leftarrow PC_2(C_i, D_i)$;
end
return $K_1, \ldots, K_{16}$;

Output bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Input bit	16	25	12	11	3	20	4	15	31	17	9	6	27	14	1	22
Output bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input bit	30	24	8	18	0	5	29	23	13	19	2	26	10	21	28	7

We now focus our attention on the key schedule. The 56-bit key is given as a master key K of 64 bits, where the bits $K[8i]$ are used as parity checks. This key is loaded into two 28-bit registers (C_0, D_0) using a fixed permutated choice PC_1. Since this permutated choice is irrelevant to cryptanalysis, we will assume $K = (C_0, D_0)$ in the rest of this text. For each round, both registers are rotated to the left by either 1 or 2 bits, an operation denoted by $LS_{p(i)}$. The number of bits $p(i)$ is given by the table:

Round i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$p(i)$	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

We note that by the last round we have returned to the same value as the initial registers (C_0, D_0). This means that DES decryption can be implemented by making a very slight modification to the key schedule implementation. The round subkeys K_i are extracted from (C_i, D_i) using a fixed permutated choice PC_2:

Output bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24
Input bit	42	39	45	32	55	51	53	28	41	50	35	46	33	37	44	52	30	48	40	49	29	36	43	54
Output bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input bit	15	4	25	19	9	1	26	16	5	11	23	8	12	7	17	0	22	3	10	14	6	20	27	24

The DES would see widespread adoption in all sorts of applications in the decades following its standardisation. However, the relatively small block and key lengths were already a concern upon its proposal [DH77], and by the mid-90s it was considered that its security was severely lacking. In addition, several statistical attacks on the full specification such as Biham and Shamir's differential cryptanalysis [BS92] and Matsui's linear cryptanalysis [Mat93, Mat94a] also eroded confidence in its security. This encouraged NIST to launch a selection process to designate a successor to the DES featuring larger block and key sizes, and whose design would acknowledge the recent advances in cryptanalysis. The designated winner of this process would be a proposal by Daemen and Rijmen called Rijndael [DR99], which would become the Advanced Encryption Standard or AES [AES01] in 2001.

1.3 Symmetric Cryptanalysis

This section will discuss some basic aspects of the cryptanalysis of block ciphers, as the topics covered by several chapters of this manuscript fall under this category. We will start by providing a rough

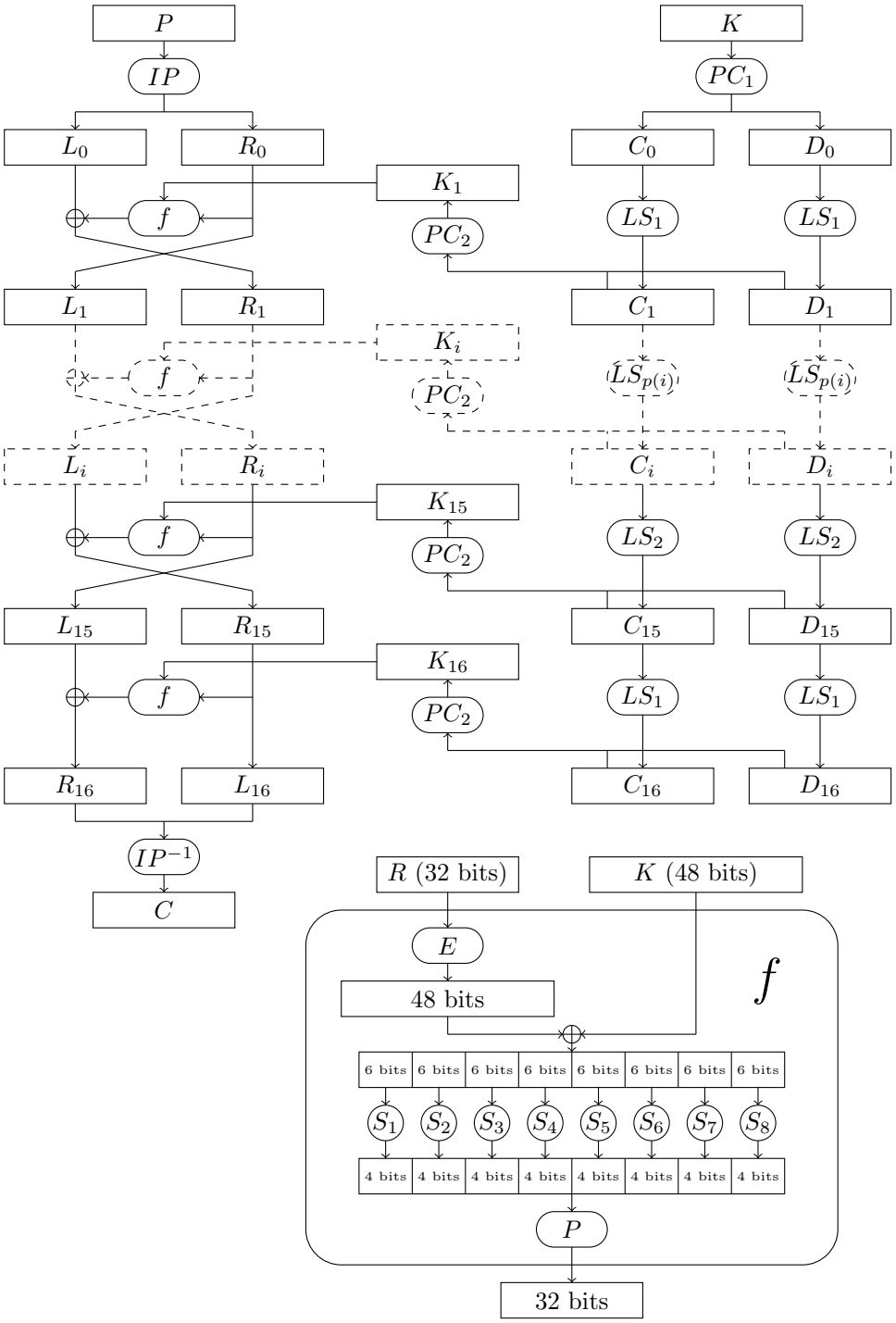


Figure 1.8: The Data Encryption Standard, its key schedule and the internal round function f.

classification of attacks on block ciphers. The section continues with a very brief overview of one of the most widely-studied families of attacks on block ciphers and other symmetric primitives in general: differential cryptanalysis. We also discuss a variant of the differential attack in the form of boomerang and rectangle distinguishers. Chapter 2 consists of a much more in-depth discussion of another well-known family of attacks: linear cryptanalysis, as it is the subject of a substantial part of my work.

1.3.1 Attack Scenarios

When the first cryptanalysts attempted to break early forms of encryption, their main objective was to extract information from some ciphertext which was intercepted in the real world. Modern symmetric cryptography, however, relies on the work of cryptanalysts in order to accurately assess the security of most primitives. This constitutes a major shift in perspective regarding the objective of cryptanalysis: while in the past most attacks had to be feasible in practice, modern cryptanalysts often consider attacks which would be completely impractical with current equipment. In general, we tend to consider attacks in scenarios which are much broader than what is actually possible to execute in the real world. This aims to create a reasonably large gap between the maximum capabilities of real-world adversaries and the capabilities which would actually be needed in order to attack the construction.

In attacks on block ciphers, we often consider that the key K is fixed and unknown to the adversary. We can classify these attacks according to which kind of *data* is available:

- **Known ciphertext attacks:** The attacker only has access to a collection of ciphertexts and some very limited information about the associated plaintexts (for example, that they consist of ASCII characters, or that they are formatted in some other specific way). The data complexity is the number of available ciphertexts.
- **Known plaintext attacks:** The attacker has obtained a (presumably random) collection of pairs of plaintexts and ciphertexts which they can analyse. We distinguish the subcategory of *distinct* known plaintext attacks, for which we assume that no two pairs are identical. The data complexity is the number of available plaintext-ciphertext pairs.
- **Chosen plaintext attacks:** The attacker has access to an encryption oracle. They can query this oracle with a plaintext x and it returns its ciphertext $E_K(x)$ under the key K. We distinguish between *non-adaptive* and *adaptive* chosen plaintext attacks. In the former, the adversary can only query a list of plaintexts and cannot make any more queries once they receive the responses. In the latter, the adversary can query plaintexts individually and formulate subsequent queries according to the given responses. The data complexity is the number of queries to the oracle.
- **Chosen ciphertext attacks:** The attacker has access to a decryption (and usually also encryption) oracle. We also separate adaptive and non-adaptive chosen ciphertext attacks. The data complexity is also the total number of queries to the oracle.
- **Related-key attacks:** In this variant of the previous attack scenarios, the oracle also accommodates related-key queries. Usually, the attacker can query encryptions or decryptions with the key $K \oplus \Delta$, where K is the secret key and Δ is a difference chosen by the attacker.

We can also consider different objectives for the attacker:

- **Distinguishing attacks:** The adversary has access to either a list of data or an encryption or decryption oracle. These may correspond to either a keyed instance of E for some unknown key K or a randomly sampled permutation $P : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$. The attacker tries to determine which of the two options is correct.
- **Target encryption/decryption:** The attacker is given a plaintext or a ciphertext and is tasked with encrypting or decrypting it under the unknown key K. The attacker has access to either a list of data or an oracle of E_K or E_K^{-1} which can be queried for any input other than the target.
- **Key recovery:** The attacker has access to either a list of data or an encryption or decryption oracle under the unknown key K and aims to determine this secret key.

Each one of these objectives is stricter than the previous one, in the sense that, for example, a key recovery attack can distinguish a block cipher from a random permutation. Some intermediate attacks may also exist. For example, an attack may construct an alternative implementation of the maps E_K or E_K^{-1} which can encrypt or decrypt any plaintext or ciphertext without explicitly recovering K.

We also need to define a baseline as to what constitutes an effective attack or "break". Most modern ciphers have concrete security claims which provide a clear threshold as to what constitutes an attack. A generic key recovery attack exists which can be applied to all block ciphers, called the exhaustive search or brute-force attack. It makes use of a single plaintext-ciphertext pair, and finds the key in at most 2^κ encryptions, where κ is the number of bits of the key. The attack consists of encrypting the plaintext under all the possible values of the key until the correct associated ciphertext is generated. For this reason, it is common to compare the time complexity of attacks on block ciphers to that of exhaustive search. This comparison is often facilitated by expressing the time complexity in terms of equivalent encryptions, that is, we compare the cost of the attack in binary operations with the cost of performing 2^κ full block cipher encryptions.

A common security claim is that the block cipher does not exhibit any effective distinguishing attacks in an adaptive chosen ciphertext scenario which have time complexity lower than that of exhaustive search, that is, 2^κ equivalent encryptions. If such a *shortcut attack* exists, we say that the cipher is *broken*, in the sense that its real security does not match its parameters. Please note that a cipher may be considered broken even if no practically feasible attack exists. Since most block ciphers are based on the iteration of rounds, we can also consider as an attack target a variant of the block cipher but with a smaller number of rounds than the full specification. We expect a smaller number of rounds to be less secure and therefore easier to attack. The difference between the maximum number of rounds on which a shortcut attack is known and the total number of rounds is called the *security margin*. In addition, in many applications, resistance against related-key attacks is also expected.

There is an additional parameter which must be considered about attacks on block ciphers, and that's to attempt to measure their effectiveness. This is very important when it comes to comparing attacks, as many attacks exhibit a trade-off between their effectiveness and their data and time complexities. In the case of key-recovery attacks, the effectiveness can be measured by estimating the probability of success P_S, that is, the probability that the attack recovers the correct key K. In the case of distinguishing attacks, the probability of success is not such a useful measure, as a coin flip which decides between both options already succeeds with probability 50%. For this reason, an approach based on hypothesis testing is used:

Definition 1.3 (Distinguishing advantage). *We consider an attack which tries to distinguish between a keyed instance of a block cipher E_K and an unknown random permutation P. The attack's* distinguishing advantage *is the difference between the probabilities of a true positive (the block cipher is correctly identified) and a false positive (the permutation is mistaken for the block cipher):*

$$\text{adv} = \Pr\left(\textit{Attack outputs "}E_K\textit{"}|E_K\right) - \Pr\left(\textit{Attack outputs "}E_K\textit{"}|P\right). \tag{1.3}$$

1.3.2 Differential Cryptanalysis

Differential cryptanalysis [BS90] was first introduced to the public by Biham and Shamir in 1990, and would become the first of a long series of statistical attacks on block ciphers. The core idea of differential cryptanalysis is to exploit XOR differences between pairs of states (or other fixed shifts by an element of a group) which propagate with high probability through the internal components of the block cipher. This section aims to provide the fundamental background necessary to describe the differential-linear and the rectangle attacks which appear later in this manuscript.

Differential cryptanalysis constitutes a very prolific family of attacks on multiple block ciphers and other constructions. Multiple variants of the differential attack exist, such as impossible differential attacks [Knu98] and truncated and higher-order differentials [Knu94], as well as boomerang [Wag99] and rectangle [KKS00, BDK01a] attacks, which are discussed in Subsection 1.3.2.3. There is a large collection of examples of impossible, truncated and classical differential attacks, including attacks on FEAL [BS91, AKM97], the DES [BS92], SAFER [KB96], CRYPTON [SK99], Camellia [LHL$^+$01, KM01, HSK02], TEA and XTEA [MHL$^+$02], Skipjack [BBS05] and CLEFIA [TTS$^+$08].

1.3.2.1 Differentials and Differential Characteristics

Definition 1.4 (Differential). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a vectorial Boolean function. A* differential *is any pair of input and output differences $\Delta, \Delta^* \in \mathbb{F}_2^n$, often denoted by $\Delta \overset{f}{\mapsto} \Delta^*$ or simply $\Delta \mapsto \Delta^*$ if no confusion is possible. The* differential probability *of the differential $\Delta \mapsto \Delta^*$ is*

$$\text{DP}(\Delta \overset{f}{\mapsto} \Delta^*) = \frac{1}{2^n} \left|x \in \mathbb{F}_2^n : f(x) + f(x+\Delta) = \Delta^*\right|. \tag{1.4}$$

In other words, the differential probability is the probability that two inputs $x, y \in \mathbb{F}_2^n$ whose difference is $\Delta = x + y$ will lead to outputs which exhibit a difference of Δ^*, that is, $f(x) + f(y) = \Delta^*$. Such input pairs are often called *good pairs* in the literature. For a random permutation, the expected value of a non-trivial differential probability is $1/2^n$, since we expect the possible output differences to be evenly distributed. This means that, if $\mathrm{DP}(\Delta \overset{f}{\mapsto} \Delta^*)$ is significantly larger than 2^{-n}, and given a collection of around $1/\mathrm{DP}(\Delta \overset{f}{\mapsto} \Delta^*)$ pairs of inputs which exhibit a difference of Δ and their corresponding outputs, it should be possible to distinguish f from a random permutation by counting the number of times that the output difference Δ^* holds. From the definition, it is clear that if f is bijective, then $\mathrm{DP}(\Delta \overset{f}{\mapsto} \Delta^*) = \mathrm{DP}(\Delta^* \overset{f^{-1}}{\longmapsto} \Delta)$.

This definition can be applied to any keyed instance of a block cipher. Since the probability of the differential might then be dependent on the key K, the definition can be expanded to the average differential probability over the keyspace, which is also the overall probability that the differential holds when both the key and the plaintext are uniformly distributed. A differential of E with high probability results in a chosen plaintext distinguishing attack.

We note that, if $L \in \mathrm{GL}(\mathbb{F}_2^n, \mathbb{F}_2^n)$ is a linear map, then the input difference Δ will propagate to the difference $L(\Delta)$ with probability 1 due to the definition of a linear map, as $L(x + \Delta) = L(x) + L(\Delta)$. In addition, adding a fixed constant, for example xoring a round subkey, does not change the difference Δ between two states.

Given a target block cipher, we may wish to estimate the differential probability of a given differential, or to search for differentials of high probability. In the case of key-alternating block ciphers, we can describe the differential probability of a differential $\Delta \mapsto \Delta^*$ as the sum of the probabilities of all the possible propagations of the difference which start at Δ and end at Δ^*.

Definition 1.5 (Differential characteristics). *Given a function $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ which is the composition of r round functions $F_1, \ldots, F_r$, $F = F_r \circ \cdots \circ F_1$, a* differential characteristic *is an $(r+1)-$tuple of differences $(\delta_0, \ldots, \delta_r)$, $\delta_i \in \mathbb{F}_2^n$. The differential probability of a differential characteristic is the probability that the difference between the states for a pair of plaintexts with input difference δ_0 is equal to δ_i for every round:*

$$\mathrm{DP}(\delta_0, \ldots, \delta_r) = \frac{1}{2^n} \left| x \in \mathbb{F}_2^n : \begin{cases} F_1(x) + F_1(x + \delta_0) = \delta_1, \\ (F_2 \circ F_1)(x) + (F_2 \circ F_1)(x + \delta_0) = \delta_2, \\ \qquad\qquad \vdots \\ (F_r \circ \cdots \circ F_1)(x) + (F_r \circ \cdots \circ F_1)(x + \delta_0) = \delta_r \end{cases} \right. . \tag{1.5}$$

Under certain independence assumptions [LMM91], the total differential probability is the product of the differential probabilities for each of the functions in the composition:

$$\mathrm{DP}(\delta_0, \ldots, \delta_r) = \prod_{i=1}^{r} \mathrm{DP}(\delta_{i-1} \overset{F_i}{\longmapsto} \delta_i). \tag{1.6}$$

The independence assumptions are actually quite strict, as in normal circumstances it may be the case that the probability that a round differential $\delta_{i-1} \overset{F_i}{\longmapsto} \delta_i$ is satisfied conditioned to the assumption that all the previous differentials were satisfied may be very different from $\mathrm{DP}(\delta_{i-1} \overset{F_i}{\longmapsto} \delta_i)$. In other words, the fact that a certain pair of inputs satisfied the first $i-1$ differential transitions may have an influence on the probability that it satisfies the i-th. In the case of key-alternating block ciphers, this situation is somewhat alleviated because we expect the key addition to randomise the inputs to each individual round: although the differential behaviour of each of the rounds may not be independent for a fixed key, we consider it is safe to assume independence under a random uniformly distributed key.

Proposition 1.6. *Let $F = F_r \circ \cdots \circ F_1 : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a vectorial Boolean function which is the composition of r round functions, and let $\Delta, \Delta^* \in \mathbb{F}_2^n$ be input and output differences. The differential probability of $\Delta \overset{F}{\mapsto} \Delta^*$ is:*

$$\mathrm{DP}(\Delta \overset{F}{\mapsto} \Delta^*) = \sum_{\delta_1 \in \mathbb{F}_2^n} \cdots \sum_{\delta_{r-1} \in \mathbb{F}_2^n} \mathrm{DP}(\Delta = \delta_0, \delta_1, \ldots, \delta_{r-1}, \delta_r = \Delta^*). \tag{1.7}$$

This result can be used to compute the differential probability of any differential of a block cipher: it suffices to consider all possible intermediate differences on each round of the cipher, and add up the differential probabilities of the associated differential characteristics. Since the number of differential characteristics is often extremely large, it is common to ignore all characteristics with differential probability lower than a certain threshold and consider their contribution negligible. Depending on the distribution

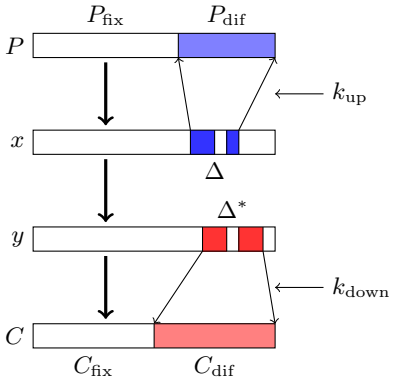


Figure 1.9: Schematic of a simple differential key recovery attack.

of the differential probabilities of the characteristics, this can lead to reasonably accurate results. In some cases, there's a single differential characteristic which dominates the probability of the whole differential.

A common extension of classical differential cryptanalysis is the use of *truncated differentials* [Knu94]. Instead of determining fixed input and output differences Δ and Δ^*, broader sets of valid input and output differences are considered:

Definition 1.7 (Truncated differential). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a vectorial Boolean function. A* truncated differential *of f is a pair of subsets $\overline{\Delta}, \overline{\Delta^*} \subseteq \mathbb{F}_2^n$ of valid input and output differences, and is denoted by $\overline{\Delta} \overset{f}{\mapsto} \overline{\Delta^*}$. The differential probability of $\overline{\Delta} \mapsto \overline{\Delta^*}$ is*

$$\mathrm{DP}\left(\overline{\Delta} \overset{f}{\mapsto} \overline{\Delta^*}\right) = \frac{\left|x, y \in \mathbb{F}_2^n : x + y \in \overline{\Delta} \text{ and } f(x) + f(y) \in \overline{\Delta^*}\right|}{\left|x, y \in \mathbb{F}_2^n : x + y \in \overline{\Delta}\right|}. \tag{1.8}$$

For a random permutation, the differential probability of a truncated differential $\overline{\Delta} \overset{f}{\mapsto} \overline{\Delta^*}$ is $\left|\overline{\Delta^*}\right| / 2^n$, which means that it can be used to distinguish f from a random permutation using around $\left|\overline{\Delta^*}\right| / \mathrm{DP}\left(\overline{\Delta} \mapsto \overline{\Delta^*}\right)$ input pairs. The actual number of required queries to f may be much lower depending on the structure of $\overline{\Delta}$. Following the template of this definition, we can define truncated differentials of block ciphers as well as truncated differential characteristics. This extension to the theory can be used in order to obtain differentials of higher probability, and is a useful way of describing the behaviour of standard differentials.

Normally, truncated differentials consider subsets which consist of fixed differences in some parts of the state and undetermined differences in other parts of the state. This corresponds to the idea of letting the output difference of some specific Sboxes be undetermined. For example, if the state consists of four nibbles, the truncated difference $\overline{\Delta} = \texttt{0??A}$ means that the difference in the leftmost nibble is zero, the difference in the rightmost nibble is exactly `A`, and the differences in the two middle nibbles can take any value. This truncated difference is thus an affine subspace of $\mathbb{F}_2^{16}$ of dimension 8.

1.3.2.2 Differential Key Recovery Attacks

Differential cryptanalysis is often applied to block ciphers by means of a key recovery attack, such as Biham and Shamir's differential attack on the DES [BS92]. In it, a differential $\Delta \mapsto \Delta^*$ is extended for a few rounds by adding some key guesses. For each key guess in the first few rounds of the cipher, the attacker makes encryption queries for pairs of plaintexts which are carefully chosen so that they will show the input difference Δ at the start of the differential. Similarly, the key guess at the bottom is used to check whether the output difference Δ^* is verified. If enough pairs are generated for each key guess, then good differential pairs should appear for the guess of the key corresponding to the secret key K with a much larger probability than for the other guesses. The rest of the key can be found through an exhaustive search. From an algorithmic point of view, we have to handle the key guesses on the plaintext side, which influence the pair generation, and the key guesses on the ciphertext side, which influence the pair checking.

We first focus on the checking of the pairs, assuming there is no key recovery at the top part. The output difference Δ^* of the differential propagates through the last rounds of the cipher and produces an undetermined difference in some parts of the ciphertext C_{dif}, while the rest of the ciphertext C_{fix} is unaffected. Given a ciphertext pair, we need to guess the parts of the key k_{down} which interact with C_{dif} in order to partially invert the last few rounds and find whether the output difference Δ^* holds. We note

that if a difference between the ciphertexts is present in C_{fix}, we can automatically reject the pair for all key guesses. Once we have removed all these rejected pairs, we can find the key guesses which result in the output difference Δ^* for each of the remaining pairs. We count how many good pairs exist for each key guess k_{down}.

The pair generation is a bit more complicated. The input difference Δ propagates backwards into some undetermined difference in the plaintext which occupies a part P_{dif}, while the rest of the plaintext P_{fix} should be identical in both elements of a plaintext pair. In theory, we could just guess each individual value of a part of the key k_{up} separately, and ask the oracle for plaintext pairs which will generate the appropriate difference Δ under that specific key guess. This, however, would result in a great increase in both data and time complexity. This problem is solved by using *plaintext structures*. We choose some fixed value of P_{fix}, and ask the oracle to encrypt the plaintexts corresponding to all possible values of P_{dif}. For each guess of the key k_{up}, we can look at each plaintext in the structure and find the associated plaintext which will result in a pair with the input difference Δ. This process can be repeated for other values of P_{fix} until enough pairs are available to perform the attack.

The key guessing can be made more time efficient if we make use of the fact that all ciphertext pairs which are not equal in C_{fix} will be rejected (this is often called *early rejection*, *sieving* or *filtering*). Indeed, if we sort all the plaintexts and ciphertexts in a structure according to the value of C_{fix}, we can rapidly find all pairs of plaintexts which share the same value of C_{fix}. For each of these candidate pairs, we can see which key guesses in the top part k_{up} will produce the appropriate input difference Δ, and which guesses in the bottom part k_{down} will produce the output difference Δ^*. Each guess $(k_{\text{up}}, k_{\text{down}})$ which generates a good pair is noted down for the final exhaustive search step.

1.3.2.3 Boomerang and Rectangle Attacks

There are several variants of differential cryptanalysis, one of the most popular of which is the boomerang attack [Wag99], which was introduced by Wagner. The boomerang attack is an adaptive chosen ciphertext attack which splits a block cipher E into two parts $E_K = E_K^2 \circ E_K^1$. We consider a differential $\Delta \xmapsto{E^1} \Delta^*$ of E^1 with probability $p = \text{DP}(\Delta \xmapsto{E^1} \Delta^*)$ and a differential $\nabla^* \xmapsto{E^2} \nabla$ of E^2 with probability $q = \text{DP}(\nabla^* \xmapsto{E^2} \nabla)$. The idea of the attack is that the probabilities of these two separate differentials may be significantly larger than the probability of any differential covering the full cipher E.

The classical boomerang attack is performed as follows:

1. We query the encryption of N pairs of plaintexts (x_1, x_2) which satisfy the input difference $\Delta = x_1 + x_2$ to obtain pairs of ciphertexts $(y_1 = E_K(x_1), y_2 = E_K(x_2))$. If we denote the intermediate states between both parts of the cipher as $z_i = E_K^1(x_i)$, then we know that around pN of the queried pairs will satisfy the relationship $z_1 + z_2 = \Delta^*$.

2. For each of the ciphertext pairs generated in the previous step, we query the decryption of the ciphertexts $(y_3 = y_1 + \nabla, y_4 = y_2 + \nabla)$ to obtain two new plaintexts (x_3, x_4). From the probability of the second differential, we know that around qN of these quartets will satisfy the property $z_1 + z_3 = \nabla^*$ and around qN will satisfy $z_2 + z_4 = \nabla^*$, where $z_i = (E_K^2)^{-1}(y_i)$. Furthermore, under some independence assumptions, we know that pq^2N quartets will satisfy $z_1 + z_2 = \Delta^*$ and $z_1 + z_3 = z_2 + z_4 = \nabla^*$ at the same time. By adding these relations, we conclude that $z_3 + z_4 = \Delta^*$ also holds for these.

3. For each of the quartets, we check whether $x_3 + x_4 = \Delta$ holds. We know that $z_3 + z_4 = \Delta^*$ with probability at least pq^2. Given the probability of the first differential, we conclude that the relation $x_3 + x_4 = \Delta$ should hold for around p^2q^2N quartets.

If $N > \mathcal{O}\left(p^{-2}q^{-2}\right)$, it is highly probable that a quartet of plaintexts satisfying the final difference $x_3 + x_4 = \Delta$ will be found. The structure of the attack is described in Figure 1.10. If $p^2q^2 > 2^{-n}$, this constitutes a distinguishing attack. We can adapt it into a key recovery attack by incorporating some key guesses at the top, at the bottom, or at both sides.

The estimation of the probability that the boomerang returns is somewhat inaccurate. For example, the probability may be enhanced by the fact that the middle differences Δ^* and ∇^* are not fixed, meaning that the total probability is the result of the combination of the probabilities for all possible middle differences. Furthermore, the assumption that both differentials behave independently is often false. In recent years, several tools have been proposed to better describe the behaviour at the transition between both differentials, the most well-known of which is the Boomerang Connectivity Table or BCT [CHP+18].

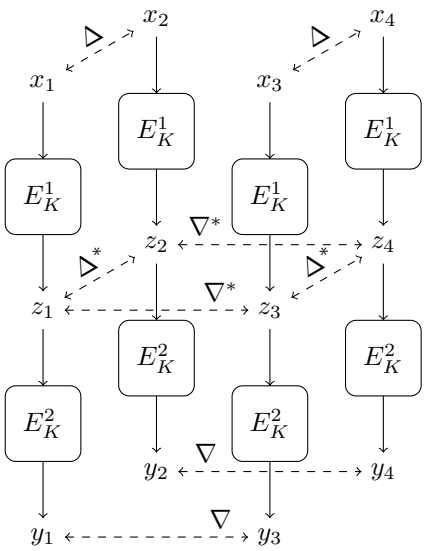


Figure 1.10: The boomerang attack.

A common variant of the boomerang attack is the so-called amplified boomerang or rectangle attack, which was introduced by Kelsey et al. [KKS00] and Biham et al. [BDK01b]. The idea of the rectangle attack is to turn the boomerang distinguisher, which is a chosen ciphertext attack, into a chosen plaintext attack, by targetting the output difference ∇ instead of the input difference Δ.

We query the encryption oracle with M plaintext pairs $(x_1, x_2 = x_1 + \Delta)$. From these we can build $\binom{M}{2}$ pairs of pairs $((x_1, x_2), (x_3, x_4))$ in total. Our aim is to find one which satisfies $y_1 + y_3 = y_2 + y_4 = \nabla$, as in that case all the differences in the boomerang distinguisher will be satisfied. Since among the M pairs, there will be around pM which satisfy $z_1 + z_2 = \Delta^*$, there are around $\binom{pM}{2} \simeq p^2M^2$ quartets (x_1, x_2, x_3, x_4) for which the differences $x_1 + z_2 = z_3 + z_4 = \Delta^*$ hold. Among these quartets, the difference $z_1 + z_3 = \nabla^*$ will happen with probability 2^{-n}. In this case, we also have $z_2 + z_4 = \nabla^*$, and will find that $y_1 + y_3 = y_2 + y_4 = \nabla$ with probability q^2. By the end, we are left with a total number of quartets equal to around $p^2 \cdot M^2 \cdot 2^{-n} \cdot q^2$. This means that we can find one such quartet if $M > \mathcal{O}\left(p^{-1}q^{-1}2^{-n/2}\right)$.

There are many examples of boomerang and rectangle attacks on block ciphers and reduced-round variants in the literature, such as COCONUT98 [Wag99, BDK05a], Serpent [KKS00, BDK01b, BDK02b], IDEA [BDK05a] and KASUMI [BDK05b].

Chapter 2

Linear Cryptanalysis

The purpose of the present chapter is to make the reader familiar with some of the most commonly-used concepts in the theory of linear cryptanalysis, as they are used frequently in this thesis. A somewhat historical approach to the introduction of the concepts has been favoured in order to better place the explanations within the context of the bibliography.

Contents

Section 2.1 provides a thorough description of "classical" linear cryptanalysis as introduced by Matsui in [Mat93, Mat94a, Mat94b], covering Algorithms 1 and 2, as well as the piling-up lemma and the famous attack on the DES. These topics provide a basis which justify the motivations for the subjects which are discussed in the rest of this chapter. Section 2.2 provides a more detailed analysis of the correlation of linear approximations, including the concepts of linear hulls and linear trails [Nyb94]. Section 2.3 covers the techniques which allow multiple linear approximations to be exploited, specifically multiple [BCQ04], multidimensional [CHN08, HCN08] and conditional [BP18] linear cryptanalysis. Section 2.4 is intended as a practical guide for the computation of the data complexity of linear key recovery attacks, and is based on the models introduced in [Sel08, BN15, BN16, BN17]. Finally, Section 2.5 provides a short introduction to differential-linear attacks [LH94, BDK02a].

2.1 Matsui's Linear Cryptanalysis

In its most basic and broadest form, linear cryptanalysis can be described as a family of attacks on symmetric constructions which make use of linear combinations of bits which are probabilistically unbalanced. This idea can be traced back to some known plaintext attacks on the block cipher FEAL [SM87] in works such as [TG91, MY92], as well as (fast) correlation attacks on stream ciphers [Sie84, MS89]. The two fundamental algorithms in linear cryptanalysis, Algorithm 1 and Algorithm 2, as well as the term linear cryptanalysis itself, were first introduced by Matsui [Mat93], who described theoretical attacks on the then-widely-used Data Encryption Standard (DES) [DES77]. Matsui would later improve on the Algorithm 2 version of the attack to perform the first experimental attack on the DES [Mat94a]. This subsection will discuss the fundamental results of these papers, as they constitute the foundation on which the theory of linear cryptanalysis would develop.

Algorithms 1 and 2 are both known plaintext attacks which exploit the statistical biasedness of a linear combination of bits. Algorithm 1 uses a linear combination of bits of the plaintext, ciphertext and key directly to recover a single keybit, while Algorithm 2 makes a partial key guess in order to use an approximation between two internal states of the cipher.

Definition 2.1 (Keyed linear approximation). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher. A* (keyed) linear approximation *of E is a specific linear combination of some bits from the plaintext, ciphertext, and key which can be written in the form*

$$\nu : \langle \alpha, x \rangle + \langle \beta, y \rangle + \gamma(K), \tag{2.1}$$

where x denotes the plaintext, y denotes the ciphertext, and K denotes the key. The vectors $\alpha, \beta \in \mathbb{F}_2^n$ are called input *and* output masks*, and the map $\gamma : \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2$ is the key mask.*

The key mask γ is written here as a potentially non-linear map from the keyspace to $\mathbb{F}_2$ because the key schedule may operate non-linearly on the key. However, in many cases it is possible to write $\gamma(K)$ as a linear combination of bits of the round subkeys.

Matsui's attacks use a *biased*, *unbalanced* or *correlated* linear approximation. This means that, if the approximation is regarded as a random variable over the key and the plaintext, the probability that it is equal to 0 is significantly different from 1/2.

Definition 2.2 (Bias and correlation of a linear approximation). *Given a linear approximation ν of a block cipher, its* bias *is the quantity*

$$\varepsilon = \Pr_{x,K}(\nu(x, E_K(x), K) = 0) - \frac{1}{2}. \tag{2.2}$$

A related magnitude is the correlation *[DGV94]*

$$c = \Pr_{x,K}(\nu(x, E_K(x), K) = 0) - \Pr_{x,K}(\nu(x, E_K(x), K) = 1). \tag{2.3}$$

Note that $-\frac{1}{2} \leq \varepsilon \leq \frac{1}{2}$ and $-1 \leq c \leq 1$. Furthermore, it is clear from the definition that the bias and the correlation of an approximation follow the relationship

$$c = 2\varepsilon. \tag{2.4}$$

In [Mat93], all the results are given in terms of the bias ε of the approximation, while modern literature tends to mostly use the correlation c for reasons which will soon become apparent. Since both quantities only differ by a constant factor this is not a major issue, however, when making calculations one should be attentive as to which measure of imbalance is being used.

Algorithm 3: Matsui's Algorithm 1

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs of a block cipher, a linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle + \gamma(K)$ with bias ε.
Output: A probable value for $\gamma(K)$.
$T \leftarrow 0$;
forall $(x, y) \in \mathcal{D}$ **do** `// Compute the counter` T
  if $\langle \alpha, x \rangle + \langle \beta, y \rangle = 0$ **then** $T \leftarrow T + 1$;
end
switch (T, ε) **do** `// Compare` T `and` $N/2$
  case $T > N/2, \varepsilon > 0$ **do return** $\gamma(K) = 0$;
  case $T > N/2, \varepsilon < 0$ **do return** $\gamma(K) = 1$;
  case $T < N/2, \varepsilon > 0$ **do return** $\gamma(K) = 1$;
  case $T < N/2, \varepsilon < 0$ **do return** $\gamma(K) = 0$;
end

2.1.1 Matsui's Algorithm 1

Matsui [Mat93] proposed two different key recovery attack algorithms which make use of a single linear approximation, Algorithms 1 and 2. Algorithm 1 uses an approximation of the whole cipher to recover one bit of information about the key. We suppose an attacker has access to a collection $\mathcal{D} = \{(x, y = E_K(x))\} \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^n$ of N known plaintext-ciphertext pairs of a block cipher E which have been generated using a fixed key K which is unknown to the attacker. We also assume they have found a linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle + \gamma(K)$ with known bias ε and correlation c.

In this situation, and under some assumptions which will be discussed shortly, Matsui's Algorithm 1 (written in detail as Algorithm 3) can recover one bit of information about K. The attacker first computes the counter

$$T = |(x, y) \in \mathcal{D} : \langle \alpha, x \rangle + \langle \beta, y \rangle = 0| \,, \tag{2.5}$$

which, for a large enough random collection of plaintext-ciphertext pairs, is expected to be approximately $N/2 \pm N\varepsilon$, with the sign depending on the value of $\gamma(K)$. By comparing this number to $N/2$, and assuming the sample of plaintext-ciphertext pairs is large enough, this bit of information about the key can be recovered with high probability.

2.1.1.1 Data and Time Complexities of Matsui's Algorithm 1

Although the original publication [Mat93] is somewhat vague and does not explicitly discuss the underlying assumptions on the behaviour of linear approximations which are used to prove its results on the data complexity, it is possible to formalise the intuitive idea behind Algorithm 1 as the following statement (see for example [BT13]):

Assumption 2.3 (Right-key equivalence hypothesis). *The key K has no influence on the correlation of $\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle$ (the linear approximation when the key part is removed):*

$$\Pr_x(\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle = 0) \simeq \begin{cases} \Pr_{x,\widetilde{K}}(\langle \alpha, x \rangle + \langle \beta, E_{\widetilde{K}}(x) \rangle + \gamma(\widetilde{K}) = 0) \text{ if } \gamma(K) = 0 \\ \Pr_{x,\widetilde{K}}(\langle \alpha, x \rangle + \langle \beta, E_{\widetilde{K}}(x) \rangle + \gamma(\widetilde{K}) = 1) \text{ if } \gamma(K) = 1 \end{cases}, \tag{2.6}$$

where $\widetilde{K}$ denotes a random key which may be different from K, which we consider fixed.

Let us briefly discuss this assumption in a little more detail. It describes the behaviour of the linear approximation once we remove the key term, and has the following consequences:

- The absolute value of the correlation of the keyless approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle$ is constant throughout the keyspace, and is thus independent of the specific value of K.
- Given the masks α and β, there is a "dominant" key mask γ so that the correlation of the keyless approximation is equal (up to the sign) to the correlation of the same linear approximation but with the term $\gamma(K)$. In other words, all other possible key masks have little to no contribution to the correlation of the keyless linear approximation.

These properties hold in the case of Matsui's linear approximations for reduced-round DES, but they do not hold generally. Indeed, for a given linear approximation of a modern block cipher there are often

multiple different key masks, called *linear trails* [DGV94] in the case of iterative block ciphers, which show appreciable correlation. As a result, the absolute correlation tends to be key-dependent. In Section 2.2 we will discuss the properties of general keyless linear approximations in more detail through the concept of *linear hull* [Nyb94].

In any case, assuming that the right-key equivalence hypothesis holds for the approximation we are using, we can estimate the success probability (that is, the probability that the guess for $\gamma(K)$ is correct) as a function of the number of known plaintexts N and the bias ε.

Proposition 2.4 (Lemma 2 in [Mat93]). *Under the right-key equivalence hypothesis, the probability of success of Matsui's Algorithm 1 is*

$$P_S \simeq \Phi\left(2\sqrt{N}|\varepsilon|\right) = \Phi\left(\sqrt{Nc^2}\right) = \int_{-\infty}^{\sqrt{Nc^2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx. \tag{2.7}$$

Proof. Without loss of generality, we can assume that $\gamma(K) = 0$ and $\varepsilon > 0$. In this case, the right-key equivalence hypothesis implies that $\Pr_x(\langle \alpha, x\rangle + \langle \beta, E_K(x)\rangle = 0) = 1/2 + \varepsilon$. The counter T is thus a random variable which follows a binomial distribution $\mathcal{B}(1/2+\varepsilon, N)$ and can be approximated by the normal distribution $\mathcal{N}(N(1/2+\varepsilon), N/4)$. In this case, the probability of success P_S is the probability that the algorithm returns $\gamma(K) = 0$, which happens when $T > N/2$. By expressing T as $T = N/2 + N\varepsilon + (\sqrt{N}/2)X$, where the random variable X follows a standard normal distribution, we deduce:

$$P_S = \Pr\left(T > N/2\right) = \Pr\left(N\varepsilon + (\sqrt{N}/2)X > 0\right) = \Pr\left(X < 2\sqrt{N}|\varepsilon|\right) = \Phi\left(2\sqrt{N}|\varepsilon|\right). \qquad \square$$

This means around $N \sim 1/4\varepsilon^2 = 1/c^2$ known plaintext-ciphertext pairs are required for the Algorithm 1 attack to succeed with reasonable probability. After one bit of information about the key has been recovered, an exhaustive search over all the possible keys can be performed in order to retrieve the remaining bits. The time complexity of a full key-recovery attack using Algorithm 1 is therefore $O(N) + 2^{\kappa-1}$ equivalent encryptions. Alternatively, it is also possible to repeat the attack with different approximations so that more keybits are recovered.

2.1.2 Matsui's Algorithm 2

Given the limitations of Algorithm 1 (specifically the fact that it can only recover one bit of information about the key), Matsui also proposed an attack which permits the recovery of a larger part of the key while attacking a larger number of rounds. Furthermore, this attack makes use of a weaker assumption than Assumption 2.3. The "trick" consists of using a partial approximation of the cipher which covers most of the rounds, and guessing the necessary parts of the key in order to cover the rest of the cipher. Since the approximation covers a smaller number of rounds than in Algorithm 1, we can expect to find an approximation with a larger bias, which either reduces the data complexity or allows more rounds to be attacked. It has the additional advantage of recovering several bits of the key all at the same time, which reduces the time complexity of the exhaustive search step at the end of the attack.

For simplicity, we will assume only the last round is covered by the key recovery, although the same attack framework can be used when we guess key material over several rounds at both the input and output sides of the cipher. Let us split E_K as $E_K = F_K \circ E'_K$, so that F_K represents the last round, and let $\langle \alpha, x\rangle + \langle \beta, E'_K(x)\rangle + \gamma(K)$ be a linear approximation of E'_K with bias ε, that is, an approximation of the first $r-1$ rounds of the cipher.

We will also suppose that computing $\langle \beta, F_K^{-1}(y)\rangle$ from y (that is, inverting the last round and computing the associated term of the linear approximation) only requires a few bits of K, which we will denote by k. There are $2^{|k|}$ such possibilities for this segment of the key, all of which have to be considered by the attacker. Given this partial guess k and the ciphertext y, we define the truncated last round decryption $f : \mathbb{F}_2^n \times \mathbb{F}_2^{|k|} \longrightarrow \mathbb{F}_2$ as the map which verifies

$$f(y,k) = \langle \beta, F_K^{-1}(y)\rangle \text{ for all } y \in \mathbb{F}_2^n \text{ and } K \in \mathbb{F}_2^\kappa \text{ which match the guess } k. \tag{2.8}$$

Given $\mathcal{D} = \{(x, y = E_K(x))\}$ a collection of N plaintext-ciphertext pairs, the attacker can retrieve the partial key k using Matsui's Algorithm 2 (written in detail as Algorithm 4). The attacker *guesses* all the possible values of k and for each one, they compute the counter

$$T_k = \left|(x,y) \in \mathcal{D} : \langle \alpha, x\rangle \oplus f(y,k) = 0\right|. \tag{2.9}$$

The attacker then keeps the guess whose counter T_k is most different from $N/2$. For the right key guess, its expected value is $N/2 \pm N\varepsilon$. For the wrong guesses, we assume the cipher behaves like a random permutation, so the expected value will be $N/2$. The attack thus relies on the ability of the linear approximation to distinguish the cipher from a random permutation.

Algorithm 4: Matsui's Algorithm 2

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs of a block cipher, a linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle + \gamma(K)$ with bias ε.
Output: A probable guess for k.

```
T ← 0;
forall (x, y) ∈ D do                                   // Compute the counters T_k
    forall k ∈ F_2^{|k|} do
        if ⟨α, x⟩ + f(y, k) = 0 then T_k ← T_k + 1;
    end
end
return argmax_k(|T_k − N/2|);                          // Find the T_k most different to N/2
```

2.1.2.1 Data Complexity of Matsui's Algorithm 2

As with Algorithm 1, it is possible to deduce that Algorithm 2 relies on the following assumption:

Assumption 2.5 (Wrong-key randomisation hypothesis). *For all the wrong guesses of k, the correlation of the linear approximation is approximately zero. If $\tilde{k} \neq k$, then*

$$\Pr_x \left(\langle \alpha, x \rangle + f(y, \tilde{k}) = 0 \right) \simeq 1/2. \tag{2.10}$$

We also assume that the values of the linear approximation for each of the different wrong guesses $\tilde{k}$ are independent from each other.

This assumption is generally considered to be more reasonable than the right-key equivalence hypothesis, since it is justified, at least intuitively, by the idea that decrypting the last round of the cipher with the wrong key should have a similar effect to performing an additional round of encryption. If we assume both the right-key equivalence and the wrong-key randomisation hypotheses, we can approximate the probability of success of the attack:

Proposition 2.6 (Lemma 5 in [Mat93]). *Under the right-key equivalence and the wrong-key randomisation hypotheses, the success probability of Matsui's Algorithm 2 is*

$$P_S \simeq \int_{-2\sqrt{N}|\varepsilon|}^{\infty} \left(\int_{-x-2\sqrt{N}|\varepsilon|}^{x+2\sqrt{N}|\varepsilon|} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2} dy \right)^{2^{|k|}-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx. \tag{2.11}$$

Proof. Without loss of generality, we suppose that $\varepsilon > 0$ and $\gamma(K) = 0$. From the statistical assumptions, the distribution of T_k can be approximated by $\mathcal{N}(N/2 + N\varepsilon, N/4)$ if k is the right key guess, while for any wrong key guess $T_{\tilde{k}}$ follows the distribution $\mathcal{N}(N/2, N/4)$, and they are all independent. The probability of success P_S is the probability that the counter T_k for the right key is larger than $N/2$ and $|T_k - N/2|$ is larger than all the other $|T_{\tilde{k}} - N/2|$, that is:

$$\begin{aligned} P_S =& \Pr_x \left(T_k > N/2,\ |T_k - N/2| > |T_{\tilde{k}} - N/2| \text{ for all } \tilde{k} \neq k \right) \\ =& \Pr_x \left(T_k - N/2 > 0,\ T_k - N/2 > T_{\tilde{k}} - N/2,\ T_k - N/2 > N/2 - T_{\tilde{k}} \text{ for all } \tilde{k} \neq k \right). \end{aligned}$$

Using the probability density function for the normal distribution, we obtain

$$P_S = \int_0^{\infty} \left(\int_{-s}^{s} \sqrt{\frac{2}{\pi N}} e^{-\frac{2}{N}t^2} dt \right)^{2^{|k|}-1} \sqrt{\frac{2}{\pi N}} e^{-\frac{2}{N}(s - N\varepsilon)^2} ds.$$

The final expression is obtained after the change of variables $s = (\sqrt{N}/2)x + N\varepsilon,\ t = (\sqrt{N}/2)y$. □

Since computing the probability of success using the previous expression would require the use of numerical integration, the following corollary can be used instead:

Corollary 2.7 (Lemma 4 in [Mat94a]). *The probability of success of Matsui's Algorithm 2 depends solely on the quantities $|k|$ and $4N\varepsilon^2 = Nc^2$.*

Algorithm 5: Matsui's Algorithm 2, modified

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs, a linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle + \gamma(K)$ with bias ε.
Output: A probable guess for k.
$\mathbf{a}^0 \leftarrow \underline{\mathbf{0}}$; $\mathbf{a}^1 \leftarrow \underline{\mathbf{0}}$; $\mathbf{T} \leftarrow \underline{\mathbf{0}}$;
forall $(x, y) \in \mathcal{D}$ **do** $a^{\langle \alpha, x \rangle}_{y|_\chi} \leftarrow a^{\langle \alpha, x \rangle}_{y|_\chi} + 1$; `// Extract information from data`
forall $k \in \mathbb{F}_2^{|k|}$, $j \in \mathbb{F}_2^{|k|}$ **do** `// Compute the counters` T_k
  if $f(j, k) = 0$ **then** $T_k \leftarrow T_k + a^0_j$ **else** $T_k \leftarrow T_k + a^1_j$;
end
return $\text{argmax}_k(|T_k - N/2|)$; `// Find` T_k `furthest from` $N/2$

In order to compute the probability of success when an attack would be impractical to test experimentally, we can simulate a different attack with the same values $|k|$ and Nc^2, but where c^2 is much larger and therefore N can be much smaller. For instance, Matsui performed experiments on the same attack on the DES over a reduced number of rounds. In general, we see that Matsui's Algorithm 2 requires $O\left(Nc^2\right)$ plaintext-ciphertext pairs to recover the key with a reasonable probability.

As we have already discussed, when compared to Algorithm 1, Algorithm 2 has several advantages, such as allowing the attacker to use approximations with higher correlation and recovering multiple bits of the key. Furthermore, although the estimation we have provided for the success probability is still reliant on the right-key equivalence hypothesis, we will see in the upcoming sections that Algorithm 2 can still be very effective even when the right-key equivalence hypothesis is false. Broadly speaking, Algorithm 2 does not require the existence of a dominant key mask, although its probability of success can vary largely over the keyspace.

2.1.2.2 Time Complexity of Matsui's Algorithm 2

If we look at Algorithm 4, we can see that it takes $O\left(N2^{|k|}\right)$ one-round decryptions to compute all the counters T_k, and the memory required to store them is $O\left(2^{|k|}\right)$ registers. The search phase is faster than in Algorithm 1, because we already know $|k|$ bits of the key K. The time complexity of a full key recovery attack using Matsui's Algorithm 2 is thus $O\left(N2^{|k|}\right) + 2^{\kappa - |k|}$.

In [Mat94a], it was noted that in many cases we will only require a small part $y|_\chi$ of the ciphertext y to compute $f(y, k)$, where χ denotes some mask covering the relevant bits. We can reduce the truncated last round even further to a map $f : \mathbb{F}_2^{\text{wt}(\chi)} \times \mathbb{F}_2^{|k|} \longrightarrow \mathbb{F}_2$ by removing not just the irrelevant keybits, but also the irrelevant ciphertext bits. The attack can then proceed in two steps, as shown in Algorithm 5:

1. **Distillation phase:** We first count the number of occurrences of each $(\alpha \cdot x \mid y|_\chi)$ and store them in a table. This requires N parity evaluations.

2. **Analysis phase:** We compute T_k for every key guess k from the counters of the previous step, which requires $2^{\text{wt}(\chi)+|k|}$ one-round decryptions.

As a result, this technique reduces the complexity when $2^{|k|} < N$.

A popular further improvement to this version of the algorithm was introduced by Collard et al. [CSQ07b], and uses the fast Walsh transform in order to reduce the time complexity to $\mathcal{O}(N) + \mathcal{O}\left(|k|2^{|k|}\right)$. This improvement is the main focus of Chapter 5.

2.1.3 Finding Linear Approximations

Both algorithms in Matsui's linear cryptanalysis make use of a linear approximation which has a large correlation in absolute value, as the data complexity of a linear attack is inversely proportional to the square of the bias. This raises the following questions:

- Given a linear approximation of a block cipher, how can we estimate its correlation?
- How can we find linear approximations with large absolute correlation?

These questions have been the subject of a substantial amount of work which continues to this day. Broadly speaking, we want to make use of the structure of the cipher in order to deduce the behaviour

of its linear approximations from the properties of its components. In particular, in the case of iterative block ciphers, we want ways to combine linear approximations of each round, as well as easily describe the linear approximations of the round function.

2.1.3.1 The Piling-up Lemma

We will now discuss how to combine partial approximations on ciphers which consist of the iteration of a number of rounds. In particular, we will consider key-alternating ciphers (Definition 1.2), in which the key material is xored to the state between the application of round functions. Let us assume that given a single round of a key-alternating cipher, we have some way of describing the bias of its linear approximations quite easily by studying the construction of the round function. These one-round approximations can be combined to construct a composite approximation of the whole cipher whose correlation is given by the following results:

Lemma 2.8 (Piling-up lemma, [Mat93]). *If $X_1, \dots, X_r$ are independent Bernoulli variables $\mathcal{B}(1-\varepsilon_i, 1)$ whose value is 0 with probability $1/2 + \varepsilon_i$, then*

$$\Pr(X_1 + X_2 + \cdots + X_r = 0) = \frac{1}{2} + 2^{r-1} \prod_{i=1}^{r} \varepsilon_i. \tag{2.12}$$

Proof. The proof will proceed by induction on r. For $r = 1$ the equality is evident. For the general case, assuming the formula holds for $r-1$ variables, we have the following:

$$\begin{aligned}
&\Pr(X_1 + \cdots + X_r = 0) = \\
&\quad = \Pr(X_1 = 0, X_2 + \cdots + X_r = 0) + \Pr(X_1 = 1, X_2 + \cdots + X_r = 1) \\
&\quad = \left(\frac{1}{2} + \varepsilon_1\right)\left(\frac{1}{2} + 2^{r-2}\prod_{i=2}^{r}\varepsilon_i\right) + \left(\frac{1}{2} - \varepsilon_1\right)\left(\frac{1}{2} - 2^{r-2}\prod_{i=2}^{r}\varepsilon_i\right) \\
&\quad = \frac{1}{2} + 2^{r-1}\prod_{i=1}^{r}\varepsilon_i.
\end{aligned}$$

□

The Piling-up lemma is used to justify Lemma 3 in [Mat93], which has been reformulated here with modernised language:

Corollary 2.9. *Let E be an r-round key-alternating block cipher with round functions F_i, $i = 1, \dots, r$ and round subkeys K_i, $i = 0, \dots, r$. We will denote by x_i the internal state after the i-th round, that is, after applying ARK_{K_i}. We assume we have one linear approximation of each of the rounds which has the form*

$$\nu_i : \langle \alpha_{i-1}, x_{i-1} \rangle + \langle \alpha_i, x_i \rangle + \langle \alpha_i, K_i \rangle,$$

and bias ε_i. Note that the output mask of each round approximation matches the input mask of the following round, as well as the key mask. By adding these approximations we obtain

$$\langle \alpha_0, x_0 \rangle + \langle \alpha_i, x_r \rangle + \langle \alpha_0, K_0 \rangle + \cdots + \langle \alpha_{r-1}, K_{r-1} \rangle + \langle \alpha_r, K_r \rangle, \tag{2.13}$$

which is an approximation of the full block cipher E. If we assume that the round approximations are statistically independent, then its bias and correlation are

$$\varepsilon = 2^{r-1}\prod_{i=1}^{r}\varepsilon_i, \quad c = \prod_{i=1}^{r} c_i. \tag{2.14}$$

This result provides a simple way to construct approximations of the type described by Matsui and to compute their bias, under some independence assumptions. In [Mat94b], Matsui also introduced a branch-and-bound algorithm which searches for approximations with high correlation in a recursive way, which will be described in the following subsection.

The multiplicative nature of the piling-up lemma highlights some of the similarities between linear and differential cryptanalysis. The correlation of a full linear approximation is the product of the correlations of the approximations for each round, in the same way that the probability that an input difference results in the desired output difference in differential cryptanalysis is the product of the probabilities that the difference passes each round of the cipher.

Another relationship between differential and linear cryptanalysis, as noted in [Bih94], is the different ways in which differences and linear masks interact with the basic linear operations in a block cipher (that

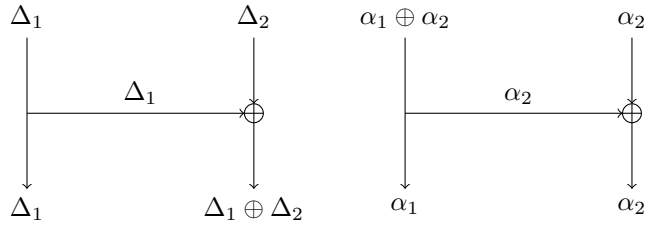


Figure 2.1: The "dual" propagation of differences and linear masks.

Algorithm 6: Matsui's Branch-and-bound Algorithm

Input: An r-round iterative block cipher E, a function $\text{COR} : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow [-1, 1]$, where $\text{COR}(\alpha, \beta)$ is the correlation of the round approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle + \langle \beta, K \rangle$.
Parameters: Correlation thresholds $(\Theta_1, \ldots, \Theta_r)$.
Output: A collection $\mathcal{S} = \{\nu = (\alpha_0, \ldots, \alpha_r; c)\}$ of approximations of E of the form $\langle \alpha_0, x \rangle + \langle \alpha_r, y \rangle + \langle \alpha_0, K_0 \rangle + \cdots + \langle \alpha_r, K_r \rangle$ and their correlations $|c| \geq \Theta_r$.

```
global S ← ∅;
forall α_0 ∈ F_2^n do
    ExtendApprox((α_0), 1, 1);
end
return S;
Procedure ExtendApprox((α_0, ..., α_{i-1}), c, i) is
    forall α_i ∈ F_2^n do
        c_new ← c · COR(α_{i-1}, α_i);
        if |c_new| ≥ Θ_i then
            if i = r then
                S ← S ∪ {(α_0, ..., α_r; c_new)};
            else
                ExtendApprox((α_0, ..., α_i), c_new, i + 1);
            end
        end
    end
end
```

is, branching points and exclusive-or operations). While differences are xored when a bitwise addition is reached, linear masks do so on branching points. Analogously, the masks for all three inputs and outputs of a XOR operation have to be identical for the correlation to be non-zero, while the same is true for differences in a branching point. For this reason, some say that the propagation of linear masks and differences are "dual" to each other.

2.1.3.2 Matsui's Branch-and-bound Algorithm

Although the piling-up lemma permits the construction and bias computation of linear approximations, on its own it is not sufficient to search for approximations of high bias, as the search space is usually insurmountably large. In [Mat94b], a heuristic depth-first branch-and-bound search algorithm was proposed, which explores the linear approximations by recursively building approximations of an increasing number of rounds, and rejecting those whose correlation is zero or falls below a certain threshold.

The underlying idea of the algorithm is that, because of the multiplicative nature of the piling-up lemma, an approximation for a small amount of rounds ν' which has a very small correlation (when compared to other approximations for the same number of rounds) is unlikely to be extended to an approximation ν which has a high correlation. This means that, if we remove from the search space all approximations ν which start with ν', it is very unlikely that any approximations of interest were removed. Furthermore, if the correlation of ν' is zero, then the correlation of any extended approximation ν is also zero.

A general version of the algorithm is depicted in Algorithm 6, and an example is shown in Figure 2.2. The algorithm starts from each possible mask α_0, and explores all the possible extensions of a given partial approximation before moving on to the next (hence the *depth-first*). The correlation of each

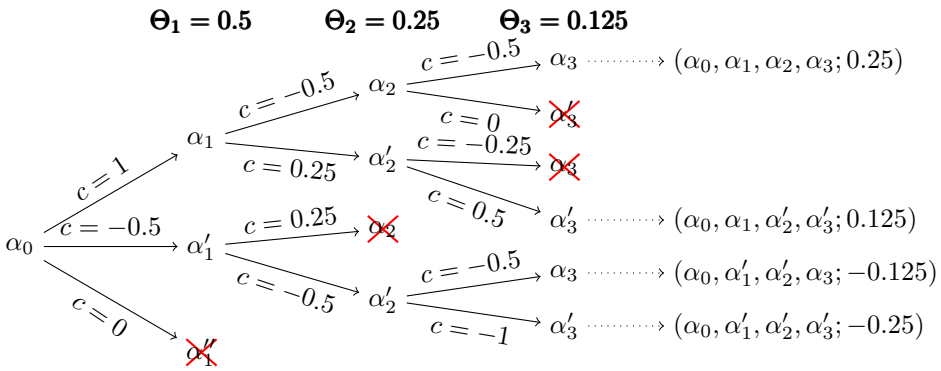


Figure 2.2: An illustration of Matsui's branch-and-bound algorithm.

.

partial approximation is computed using the piling-up lemma. However, this tree would be too large to explore in most cases, so a series of round correlation thresholds $\Theta_i : i = 1, \ldots, r$ are used as parameters. For any node corresponding to an i-round partial approximation, if its correlation falls below Θ_i, all of its children are ignored (hence the *branch-and-bound*). By the end, a list of approximations for the desired number of rounds which have correlations above Θ_r has been constructed.

The choice of the thresholds Θ_i is somewhat non-trivial, and can have a very large impact on the effectiveness and performance of the algorithm. If the values of Θ_i are too large, it is possible that some high correlation approximations are removed from the search space and missed by the algorithm, or that no approximations are found by the end at all. However, if the values of Θ_i are too small, many more approximations will be found, but the search space will be a lot larger and therefore the running time will increase. This trade-off has to be explored experimentally for each cipher until a good set of thresholds is found.

In practice, the branch and bound algorithm as described, although a lot less costly than an exhaustive search over the whole space, cannot be used without further optimisation. Indeed, the cost of exploring just the level at depth 1 is already $\mathcal{O}\left(2^{2n}\right)$ for an n-bit cipher. However, given the simple structures of most block cipher round functions, it is often possible to greatly reduce the number of masks which have to be considered to extend each partial approximation.

2.1.3.3 The Linear Approximation Table of an Sbox

In order to use the branch and bound algorithm effectively, we need ways to quickly enumerate all linear approximations of the round function of a block cipher whose correlation is above a certain threshold. Although, naturally, each typical block cipher construction will require different tools in order to achieve this, there are some which are very widely used. In this subsection, we will briefly explain one of the most commonly used constructions: the Sbox layer.

If a round of a block cipher is the composition of several transformations, we can study each one of them separately and use the piling-up lemma to obtain approximations of the whole round function. We have already discussed how linear masks propagate through linear transformations, as we can use the rules of Figure 2.1 to deduce compatible output linear masks from a given input mask, all of which will exhibit a correlation of ± 1. Furthermore, for a linear transformation given by a matrix L, a mask α propagates to a mask β with non-zero correlation if and only if $\alpha = L^T\beta$ (see Proposition 3.13. Another common occurrence in block cipher constructions is that of Sbox layers, which are maps of the form:

$$\begin{array}{rccc} f: & \mathbb{F}_2^n = \mathbb{F}_2^{dr} & \longrightarrow & \mathbb{F}_2^m = \mathbb{F}_2^{ds} \\ & (x_d|\cdots|x_1) & \longmapsto & (S_d(x_d)|\cdots|S_1(x_1)) \end{array}, \tag{2.15}$$

where $S_i : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ are arbitrary maps called Sboxes.

The behaviour of all linear approximations of an Sbox is usually described by means of its so-called Linear Approximation Table:

Definition 2.10 (Linear Approximation Table). *Let $S : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ be an Sbox. The* Linear Approximation Table *or LAT of the Sbox S is the $r \times s$ integer matrix whose entries are:*

$$\mathrm{LAT}_S(\alpha, \beta) = |x \in \mathbb{F}_2^r : \langle \alpha, x\rangle + \langle \beta, S(x)\rangle = 0| - 2^{r-1}, \ \alpha \in \mathbb{F}_2^r, \ \beta \in \mathbb{F}_2^s. \tag{2.16}$$

Table 2.1: The linear approximation table of the Sbox S_5 in the Data Encryption Standard.

β \ α	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	4	0	2	2	−2	2	0	−4	4	4	2	6	−2	−2	2	2	−2	−6	4	4	0	4	6	2	2	2	0	4	−4	−4
2	0	0	−2	−2	−2	2	−4	0	2	6	0	4	0	0	−2	−2	−2	−2	0	0	−4	0	6	−6	0	4	2	6	−2	−2	−8	8
3	0	0	2	6	0	−4	−6	2	6	−2	0	4	−2	2	0	8	0	0	−2	2	0	−4	6	−2	2	−6	−8	−4	2	6	0	−8
4	0	0	−2	−2	0	0	−2	−2	0	0	−2	6	0	0	−2	6	0	4	2	−2	0	−4	2	6	4	0	−2	−6	4	−8	−2	2
5	0	0	2	−2	2	10	−4	8	0	−4	−6	2	2	−2	4	4	−2	2	−4	4	0	4	−2	−2	−10	−2	4	0	0	0	−2	−6
6	0	0	−4	4	−2	−6	2	6	−2	−6	2	−2	4	4	0	0	−6	−2	−2	2	0	−8	4	−4	−4	4	0	0	−6	−2	−2	−6
7	0	0	0	−4	0	−4	0	0	−6	−6	2	−2	10	−10	2	2	−8	−4	−8	0	0	−8	0	4	2	−2	2	2	2	2	2	−2
8	0	0	4	0	0	0	0	−4	−2	6	2	−2	−2	−2	−2	2	0	4	4	4	−4	0	4	4	2	6	−2	6	−2	10	−2	−2
9	0	0	0	0	2	2	−2	6	2	−2	2	−2	4	0	0	4	−2	2	6	−6	−4	0	0	−4	0	8	0	8	6	−2	2	2
A	0	0	2	−2	2	−10	0	0	4	0	−2	−2	−2	−2	4	8	−2	2	4	4	4	−4	6	−6	−2	6	4	0	−4	−8	−2	−2
B	0	0	−2	6	4	0	−2	−6	−12	−4	2	2	−8	4	2	−2	−4	0	6	2	4	4	2	2	0	4	2	−2	0	−8	6	−2
C	0	0	2	−2	−4	4	−6	−2	2	2	4	0	−2	−2	−4	8	0	−8	−2	−6	0	8	2	−2	2	10	0	−4	2	2	4	−8
D	0	0	−2	−2	−2	−2	−8	0	6	−6	−4	−8	4	8	6	−6	2	−6	4	4	4	4	2	2	4	0	−2	−6	−2	2	4	0
E	0	0	0	4	−2	2	2	−6	−4	−8	−4	−4	−6	−6	−2	2	10	2	−6	−2	−4	0	0	0	−2	2	−2	−2	0	0	4	0
F	0	0	−4	−4	0	4	0	−4	4	−4	0	0	−4	0	−4	0	−20	4	0	0	0	4	0	4	−4	−4	0	0	0	4	0	−4

β \ α	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	−4	0	−2	6	2	−10	4	0	0	8	2	−2	6	6	2	2	6	−6	0	−8	4	−8	2	−2	6	−2	−8	4	−4	−4
2	0	0	−2	−2	6	2	4	0	−2	2	−4	0	4	−4	2	−6	−2	−2	0	0	−12	0	−2	−6	4	8	−10	−6	−6	2	4	−4
3	0	0	2	−2	4	0	−2	−2	−2	6	−8	4	−6	−2	−4	−4	0	0	−2	10	4	0	−6	−6	−6	2	0	−4	−2	2	−4	−4
4	0	0	−2	6	0	0	−2	6	0	0	6	6	0	0	6	6	−4	0	−2	2	−4	8	−2	−6	0	−4	2	−10	0	4	2	14
5	0	0	2	−2	6	6	0	4	4	0	6	−2	−6	−2	4	−4	−6	−2	8	0	0	−4	−2	6	−2	6	4	0	−4	4	2	6
6	0	0	−4	−4	−2	2	2	6	−6	6	6	−6	0	−8	4	4	−2	2	2	−2	4	4	8	0	4	−4	0	−8	2	−2	−2	−6
7	0	0	8	4	4	0	−4	−4	2	2	−6	6	6	2	−2	−2	−4	0	4	−4	−8	0	0	4	−2	−6	−2	−2	2	2	2	−2
8	0	0	−4	0	4	−4	4	0	2	2	6	2	−2	−2	−10	2	4	0	0	0	−4	0	4	12	−6	−2	6	−10	−6	−2	2	2
9	0	0	0	0	−6	−6	−2	6	−6	−2	2	6	−4	0	−8	4	2	−2	10	6	0	−4	−4	0	4	−4	−4	4	2	10	−2	−2
A	0	0	−6	−2	−2	2	−4	−12	4	−8	−2	−2	2	−6	0	4	2	−2	0	0	−4	−4	−2	2	−6	2	0	4	4	0	−2	6
B	0	0	6	−2	4	−8	−2	2	0	0	−2	6	−4	−8	−2	−6	0	−4	2	−10	0	0	−2	−2	0	4	2	−2	0	0	−2	−2
C	0	0	2	−2	0	0	6	2	6	−2	−8	−4	−2	−2	4	0	0	0	−2	2	−4	4	6	2	6	−2	4	0	10	2	4	0
D	0	0	−2	6	14	−2	0	0	−2	−6	4	0	4	0	−2	2	2	2	4	4	−4	4	2	2	4	0	−2	2	−2	2	−4	0
E	0	0	−16	4	2	6	−2	6	−4	0	−4	4	6	−2	2	−2	2	2	2	−2	0	−4	−4	4	−2	2	−2	−2	4	4	0	−4
F	0	0	−12	−4	0	−4	0	−4	0	0	4	4	0	4	0	−4	4	4	0	0	0	4	0	−4	0	0	−4	4	4	0	4	0

In other words, the Linear Approximation Table provides scaled versions of the bias and correlation corresponding to every possible combination of input and output masks to the Sbox. If $\mathrm{LAT}_S(\alpha, \beta) = 0$, then the associated approximation shows no correlation at all, while an entry of large absolute value corresponds to a highly-correlated linear approximation.

As an example, Table 2.1 shows the full linear approximation table of $S5$, one of the Sboxes used in the Data Encryption Standard. Matsui's linear attack makes use of the fact that $\mathrm{LAT}_{S5}(\mathtt{10}, \mathtt{F}) = -20$ (bias $-20/64$, correlation $-20/32$) and $\mathrm{LAT}_{S5}(\mathtt{10}, \mathtt{E}) = 10$ (bias $10/64$, correlation $10/32$): these are highly correlated linear approximations of the Sbox which appear multiple times in the approximation used in the attack.

Looking at Table 2.1, there are several properties which are apparent. We can see that all the entries are even, that the first row and column are mostly composed of zeroes, and that the sum of all the entries in each row and column is zero. These are in fact general properties:

Lemma 2.11. *Let $S : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ be an Sbox. We assume that for each $y \in \mathbb{F}_2^s$, the number $|x \in \mathbb{F}_2^r : f(x) = y|$ is constant (in particular, this means that S is surjective and if $r = s$, it is equivalent to S being bijective). We have:*

- $\mathrm{LAT}_S(0, 0) = 2^{r-1}$, $\mathrm{LAT}_S(\alpha, 0) = 0$ *for all* $\alpha \neq 0$, $\mathrm{LAT}_S(0, \beta) = 0$ *for all* $\beta \neq 0$.
- $\mathrm{LAT}_S(\alpha, \beta)$ *is even for all* α, β.
- $\sum_{\alpha \in \mathbb{F}_2^r} \mathrm{LAT}_S(\alpha, \beta) = \pm 2^{r-1}$ *for all* $\beta \neq 0$.
- *If $r = s$ and S is bijective, then* $\sum_{\beta \in \mathbb{F}_2^r} \mathrm{LAT}_S(\alpha, \beta) = \pm 2^{r-1}$ *for all* $\alpha \neq 0$.

Proof. The first property is clear from the definition, and true for all Sboxes.

For the second, we first note that given $\alpha, \beta \neq 0$ there are exactly 2^{r-1} elements $x \in \mathbb{F}_2^r$ for which $\langle \alpha, x \rangle = 0$. Similarly, using the balancedness assumption, we deduce that there are 2^{r-1} elements x so that $\langle \beta, S(x) \rangle = 0$. By splitting both sets into two parts according to the value of the other dot product, we deduce the following equations:

$$|x \in \mathbb{F}_2^r : (\langle \alpha, x \rangle, \langle \beta, S(x) \rangle) = (0, 0)| + |x \in \mathbb{F}_2^r : (\langle \alpha, x \rangle, \langle \beta, S(x) \rangle) = (0, 1)| = 2^{r-1},$$
$$|x \in \mathbb{F}_2^r : (\langle \alpha, x \rangle, \langle \beta, S(x) \rangle) = (0, 1)| + |x \in \mathbb{F}_2^r : (\langle \alpha, x \rangle, \langle \beta, S(x) \rangle) = (1, 1)| = 2^{r-1}.$$

By subtracting both relations we obtain:

$$|x \in \mathbb{F}_2^r : (\langle \alpha, x\rangle, \langle \beta, S(x)\rangle) = (0,0)| = |x \in \mathbb{F}_2^r : (\langle \alpha, x\rangle, \langle \beta, S(x)\rangle) = (1,1)|,$$

which implies that $|x \in \mathbb{F}_2^r : \langle \alpha, x\rangle = \langle \beta, S(x)\rangle|$ is even and thus $\mathrm{LAT}_S(\alpha, \beta)$ is also even.

We will now prove that $\sum_{\alpha \in \mathbb{F}_2^r} \mathrm{LAT}_S(\alpha, \beta) = \pm 2^{r-1}$ for all $\beta \neq 0$.

$$\begin{aligned}
\sum_{\alpha \in \mathbb{F}_2^r} \mathrm{LAT}_S(\alpha, \beta) &= \sum_{\alpha \in \mathbb{F}_2^r} \left(\sum_{x \in \mathbb{F}_2^r} \mathrm{ind}_{\langle \alpha, x\rangle = \langle \beta, S(x)\rangle}(x) - 2^{r-1} \right) \\
&= \sum_{x \in \mathbb{F}_2^r} \sum_{\alpha \in \mathbb{F}_2^r} \mathrm{ind}_{\langle \alpha, x\rangle = \underbrace{\langle \beta, S(x)\rangle}_{\text{constant}}}(x) - \sum_{\alpha \in \mathbb{F}_2^r} 2^{r-1} \\
&= \sum_{x \in \mathbb{F}_2^r \setminus \{0\}} 2^{r-1} - 2^{r(r-1)} + \begin{cases} 2^r & \text{if } \langle \beta, S(0)\rangle = 0 \\ 0 & \text{if } \langle \beta, S(0)\rangle = 1 \end{cases} \\
&= (-1)^{\langle \beta, S(0)\rangle} 2^{r-1}.
\end{aligned}$$

For a fixed $x \neq 0$, since $\langle \beta, S(x)\rangle$ is some fixed value, $\langle \alpha, x\rangle = \langle \beta, S(x)\rangle$ is a non-trivial linear equation with exactly 2^{r-1} solutions on the value α. The case $x = 0$ has to be treated separately. The last property is obtained by applying the previous one on S^{-1}. □

In Table 2.1 we can also see that $\mathrm{LAT}_{S5}(\mathtt{01}, \beta) = \mathrm{LAT}_{S5}(\mathtt{20}, \beta) = \mathrm{LAT}_{S5}(\mathtt{21}, \beta) = 0$ for all β. This is a property of the Sboxes of the DES, and it's due to the fact that for any fixed $x_0, x_5 \in \mathbb{F}_2^2$, the map $S(x_5, \cdot, \cdot, \cdot, \cdot, x_0)$ is in itself a bijective 4-bit Sbox (in other words, knowledge of these two input bits provides no information whatsoever about the output).

The correlation of a linear approximation of an Sbox layer can be computed rapidly using the linear approximation tables of the involved Sboxes as a consequence of Lemma 2.8.

Lemma 2.12. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be an Sbox layer using the Sboxes $S_i : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$. Let $\alpha = (\alpha_d | \dots | \alpha_1) \in \mathbb{F}_2^n$ and $\beta = (\beta_d | \cdots | \beta_1) \in \mathbb{F}_2^m$ be input and output masks. The bias and correlation for the linear approximation $\langle \alpha, x\rangle = \langle \beta, f(x)\rangle$ of f are:*

$$\varepsilon(\alpha, \beta) = 2^{d-1} \prod_{i=1}^{d} \frac{1}{2^r} \cdot \mathrm{LAT}_{S_i}(\alpha_i, \beta_i) \qquad c(\alpha, \beta) = \prod_{i=1}^{d} \frac{1}{2^{r-1}} \cdot \mathrm{LAT}_{S_i}(\alpha_i, \beta_i) \tag{2.17}$$

We note the following:

- When $\alpha_i = \beta_i = 0$, the Sbox doesn't have any effect on the correlation.
- When $\alpha_i \neq 0$ and $\beta_i = 0$, the correlation is automatically zero.
- For Sboxes which fulfil the balancedness assumption of the previous Lemma, when $\alpha_i = 0$ and $\beta_i \neq 0$, the correlation is automatically zero.

This essentially means that we can focus on the Sboxes for which both $\alpha_i \neq 0$ and $\beta_i \neq 0$. These are often called *active* Sboxes. We can expect the correlation of linear approximations of the round function to decrease when the number of active Sboxes increases. When implementing a branch and bound linear approximation search, it is common to limit the number of active Sboxes in each round to a small quantity, as this can greatly reduce the size of the search space and therefore decrease the time complexity without erasing any desirable approximations. This, together with the fact that the active Sboxes in a one-round extension of a given linear approximation are determined by the input mask (that is, if $\alpha_i = 0$, then we only have to consider extensions for which $\beta_i = 0$), decreases the cost of the branch and bound algorithm enough that it can be used in practice.

2.1.4 Matsui's Attack on the Data Encryption Standard

We will now briefly describe the version of Matsui's attack on the DES given in [Mat94a] as an example. The attack has a data complexity of $N = 2^{43}$ known plaintext-ciphertext pairs, and uses two 14-round linear approximations to recover a total of 26 bits of the key. We can focus on one of the two approximations, as the other one is the same approximation but flipped upside down, and has the same characteristics (this is possible because the DES is a Feistel network). As we stated in the previous chapter, we will denote by P the plaintext after IP has been applied, and by C the ciphertext before applying IP^{-1}.

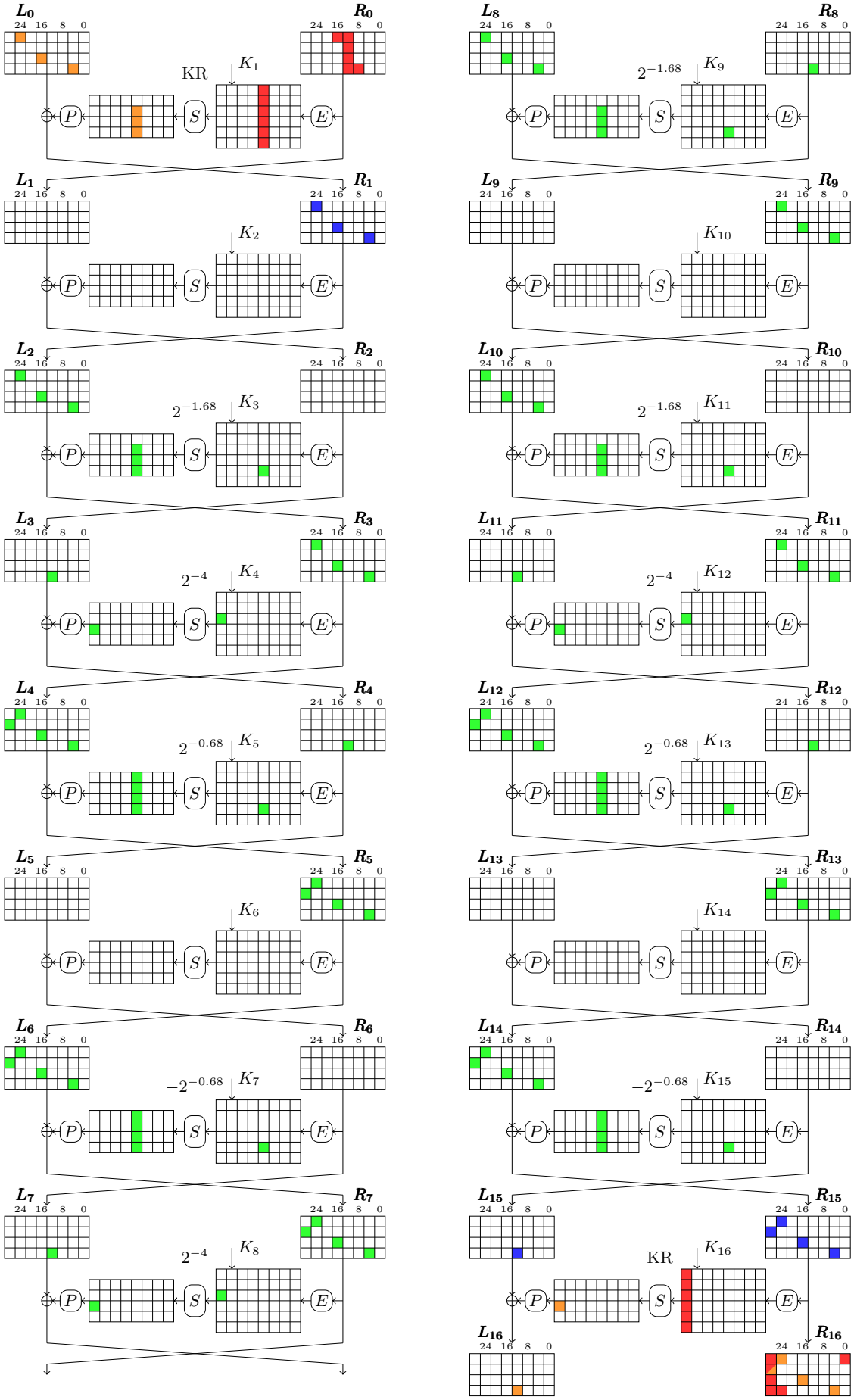


Figure 2.3: One of the two 14-round linear approximations used on Matsui's linear attack on the DES, and the associated key recovery on the first and last rounds.

Figure 2.3 depicts this 14-round linear approximation. It extends between the second and the fifteenth rounds. The input mask $\alpha = (\texttt{00000000}, \texttt{01040080})$ at (L_1, R_1) and the output mask $\beta = (\texttt{00008000}, \texttt{21040080})$ at (L_{15}, R_{15}) are highlighted in blue. All the active bits in the intermediate rounds are highlighted in light green, and the correlation for each round is given. In total, the bias of the linear approximation is computed to be $-1.9 \cdot 2^{-19}$, and its correlation is $-2^{-17.07}$. Using Algorithm 1, the parity of the sum of 10 keybits between K_2 and K_{15} can be recovered. The first and the last round are covered in a key recovery attack using Algorithm 2. In the figure, the active bits in the first and last round which appear directly in the approximation are highlighted in orange, and the bits which influence the value of the approximation but are not xored directly to it are highlighted in red. In order to compute the value of the approximation, we must guess 6 bits of K_1 and 6 bits of K_{16}. As a function of the plaintext, the ciphertext, and the key guess, the linear approximation is

$$\begin{aligned} \nu(P, C, K) = P[39, 50, 56] + \langle \texttt{E}, S5((P[16], \ldots, P[11]) + (K_1[23], \ldots, K_1[18]))\rangle \\ +C[7, 18, 24, 29, 47] + \langle \texttt{4}, S8((C[0], C[31], \ldots, C[27]) + (K_{16}[47], \ldots, K_{16}[42]))\rangle, \end{aligned} \tag{2.18}$$

where $X[i_1, \ldots, i_d]$ is a compact notation for the parity bit $X[i_1] + \cdots + X[i_d]$. The parity bit of the key which can be recovered with Algorithm 1 is

$$K_3[22] + K_4[44] + K_5[22] + K_7[22] + K_8[44] + K_9[22] + K_{11}[22] + K_{12}[44] + K_{13}[22] + K_{15}[22]. \tag{2.19}$$

Since all the 12 plaintext/ciphertext bits which appear non-linearly in the approximation are xored directly with keybits, we can define a truncated key recovery function $f : \mathbb{F}_2^{12} \longrightarrow \mathbb{F}_2$ as

$$f(x_{11}, \ldots, x_6, x_5, \ldots, x_0) = \langle \texttt{E}, S5(x_{11}, \ldots, x_6)\rangle + \langle \texttt{4}, S8(x_5, \ldots, x_0)\rangle, \tag{2.20}$$

in other words, we can obtain the desired part of the linear approximation just by feeding f the XOR of some plaintext/ciphertext bits to a 12-bit partial key guess.

The attack thus proceeds as follows:

1. **Distillation phase:**
   (a) Initialise 2^{13} counters A_x to zero.
   (b) For each of the N plaintext-ciphertext pairs (P, C), compute the 13-bit string
   $$x = (P[39, 50, 56] + C[7, 18, 24, 29, 47], P[16], \ldots, P[11], C[0], C[31], \ldots, C[27]),$$
   and increment A_x by one.
2. **Analysis phase:**
   (a) Initialise 2^{12} counters T_k to zero.
   (b) For each $k \in \mathbb{F}_2^{12}$, consider the key guess
   $$k = (K_1[23], \ldots, K_1[18], K_{16}[47], \ldots, K_{16}[42]),$$
   and let
   $$T_k = \sum_{i \in \mathbb{F}_2^{12}} (1)^{f(i+k)} \left(A_{(0|i)} - A_{(1|i)}\right).$$
   (c) Sort the T_k according to the value of $|T_k - N/2|$ and extract a list of key guesses k from the highest values. For each one, we can also obtain an additional parity bit of the key from the sign of $T_k - N/2$.
3. **Exhaustive search phase:**
   (a) Repeat the previous steps for the other approximation. We thus have two lists, each containing 13-bit candidates for part of the key. Both lists can be merged into a single list of 26-bit key candidates (see [Mat94a] for more details).
   (b) For each candidate in the merged list, test all values of the 30 remaining bits until the correct key K is found, or until all candidates are exhausted without success.

The data complexity and probability of success of the attack were obtained using Corollary 2.7. Since the data complexity mainly depends on Nc^2, an analogous attack on 8-round DES was simulated instead. This attack had a much higher value for c^2, which meant that N could be made much lower and the attack could be repeated enough times to obtain an accurate estimation of the probability of success. In the end, it was found that, given $N = 2^{43}$ plaintext-ciphertext pairs, the exhaustive search finds the correct key in less than 2^{43} test encryptions with 85% probability. The time complexity of the attack is dominated by the data generation and exhaustive search steps and reaches 2^{43} encryptions. The memory complexity is $2(2^{13} + 2^{12})$ counters. The attack was also run successfully on early 90s hardware.

2.2 Linear Approximations

The purpose of this section is to provide a more careful analysis of the behaviour of linear approximations of block ciphers. Assumption 2.3 is central to the description of linear cryptanalysis given in [Mat93], specifically in the case of Matsui's Algorithm 1, but it is quite strict. In particular, there are many instances in which a dominant key mask γ does not exist. These issues highlight the importance of a more robust analysis of the correlation of linear approximations, which can take into account the existence of multiple valid key masks, and which can describe the dependence of the correlation on the key.

For linear approximations of vectorial Boolean functions in which no key is involved, correlation can be defined as follows:

Definition 2.13 (Correlation of Boolean functions). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a Boolean function, and let $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^m$ be input and output linear masks. The correlation of the linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle$ of f is*

$$\begin{aligned} \mathrm{cor}(f; \alpha, \beta) &= \Pr_{\mathrm{x}}(\langle \alpha, x \rangle + \langle \beta, f(x) \rangle = 0) - \Pr_{\mathrm{x}}(\langle \alpha, x \rangle + \langle \beta, f(x) \rangle = 1) \\ &= \frac{1}{2^n} \left(|x \in \mathbb{F}_2^n : \langle \alpha, x \rangle + \langle \beta, f(x) \rangle = 0| - |x \in \mathbb{F}_2^n : \langle \alpha, x \rangle + \langle \beta, f(x) \rangle = 1| \right). \end{aligned} \tag{2.21}$$

2.2.1 Linear Hull of a Linear Approximation

We will start by defining the correlation of a keyless linear approximation as a magnitude which is key-dependent and may thus vary over the keyspace:

Definition 2.14. *Let $E_K : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a keyed instance of a block cipher for a specific key K. Let $\alpha, \beta \in \mathbb{F}_2^n$ be the input and output masks to the linear approximation $\nu(\alpha, \beta) : \langle \alpha, x \rangle + \langle \beta, y \rangle$. The correlation of the linear approximation for the key K is:*

$$\mathrm{cor}_K(\alpha, \beta) = \Pr_x \left(\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle = 0 \right) - \Pr_x \left(\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle = 1 \right) = \mathrm{cor}(E_K; \alpha, \beta). \tag{2.22}$$

This definition of the correlation corresponds to applying Definition 2.13 to a specific keyed instance of a block cipher is identical to the one given in Definition 2.2 but with a fixed value of the key K. This fixed-key correlation can be rewritten as:

$$\mathrm{cor}_K(\alpha, \beta) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle}. \tag{2.23}$$

This is a useful alternative definition which allows for easy formal manipulation. We will be using the following lemma, which is a preliminary version of Parseval's identity and Lemma 3.12.

Lemma 2.15. *For any fixed $x \in \mathbb{F}_2^n$, the following equality holds:*

$$\sum_{y \in \mathbb{F}_2^n} (-1)^{\langle x, y \rangle} = \begin{cases} 0 & \text{if } x \neq 0 \\ 2^n & \text{if } x = 0 \end{cases} \tag{2.24}$$

Proof. The equality is true when $x = 0$, since $(-1)^0 = 1$. When $x \neq 0$, the equality $\langle x, y \rangle = 0$ is a non-trivial linear equation over $\mathbb{F}_2^n$ which is satisfied by the elements of a hyperplane of $\mathbb{F}_2^n$, which has 2^{n-1} elements. The sum is thus $2^{n-1} \cdot (-1)^0 + 2^{n-1} \cdot (-1)^1 = 0$. □

Let us now consider keyed linear approximations. We can define a family of "linear key mask" keyed linear approximations and their correlations by using all possible linear key masks. Indeed, if $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ is a block cipher and $\alpha, \beta \in \mathbb{F}_2^n$, $\gamma \in \mathbb{F}_2^\kappa$ are linear masks, we consider the keyed linear approximation

$$\nu(\alpha, \beta, \gamma) : \langle \alpha, x \rangle + \langle \beta, y \rangle + \langle \gamma, K \rangle, \tag{2.25}$$

whose correlation can be defined as

$$\mathrm{COR}(\alpha, \beta, \gamma) = \frac{1}{2^{n+\kappa}} \sum_{K \in \mathbb{F}_2^\kappa} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle + \langle \gamma, K \rangle}. \tag{2.26}$$

The correlation of a linear approximation for any specific key can be computed by adding up the correlations of all keyed approximations. The sign of the contribution of each keyed mask is determined by the dot product of the key and the key mask:

Proposition 2.16. *Let $E_K : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a keyed instance of a block cipher, and let $\alpha, \beta \in \mathbb{F}_2^n$ be the input and output masks to a linear approximation $\langle \alpha, x \rangle + \langle \beta, y \rangle$. Then*

$$\mathrm{cor}_K(\alpha, \beta) = \sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K \rangle} \mathrm{COR}(\alpha, \beta, \gamma). \tag{2.27}$$

Proof. This result is a consequence of Lemma 2.15. Indeed, we can rewrite

$$\begin{aligned}
\sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K \rangle} \mathrm{COR}(\alpha, \beta, \gamma) &= \sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K \rangle} \left(\frac{1}{2^{n+\kappa}} \sum_{K' \in \mathbb{F}_2^\kappa} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_{K'}(x) \rangle + \langle \gamma, K' \rangle} \right) \\
&= \frac{1}{2^{n+\kappa}} \sum_{K' \in \mathbb{F}_2^\kappa} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_{K'}(x) \rangle} \underbrace{\sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K + K' \rangle}}_{0 \text{ if } K \neq K',\ 2^\kappa \text{ if } K = K'} \\
&= \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle} = \mathrm{cor}_K(\alpha, \beta).
\end{aligned}$$

□

The set of all these keyed approximations is called the linear hull of the linear approximation. The correlation of the linear approximation depends on the key: for some keys, the correlations of each of the keyed versions of the approximation interfere constructively and lead to a higher correlation. For others, the correlations interfere destructively and cancel each other out, leading to a smaller correlation.

We can measure the general usefulness of the linear approximation for linear cryptanalysis by using the average square correlation, also called *expected linear potential* or ELP:

Definition 2.17 (Linear hull, ELP). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks for a keyless linear approximation $\nu(\alpha, \beta)$. The* linear hull *of the approximation is the set of all keyed linear approximations using linear key masks $\nu(\alpha, \beta, \gamma)$. We define the expected linear potential or* ELP *of the linear approximation as the average*

$$\mathrm{ELP}(\alpha, \beta) = \frac{1}{2^\kappa} \sum_{K \in \mathbb{F}_2^\kappa} \mathrm{cor}_K(\alpha, \beta)^2. \tag{2.28}$$

The ELP is thus defined as the expected value over the keyspace of the square of the correlation of the linear approximation, a concept first introduced by Nyberg [Nyb94]. However, the ELP can also be seen as the accumulation of the correlation contributions of each element of the linear hull, as shown by the following result:

Theorem 2.18 (Linear hull theorem, Theorem 1 in [Nyb94]). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. The expected linear potential of the associated linear approximation can also be computed as:*

$$\mathrm{ELP}(\alpha, \beta) = \sum_{\gamma \in \mathbb{F}_2^\kappa} \mathrm{COR}(\alpha, \beta, \gamma)^2. \tag{2.29}$$

Proof. This result is also a consequence of Lemma 2.15:

$$\begin{aligned}
\mathrm{ELP}(\alpha, \beta) &= \frac{1}{2^\kappa} \sum_{K \in \mathbb{F}_2^\kappa} \mathrm{cor}_K(\alpha, \beta)^2 \\
&= \frac{1}{2^\kappa} \sum_{K \in \mathbb{F}_2^\kappa} \left(\sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K \rangle} \mathrm{COR}(\alpha, \beta, \gamma) \right)^2 \\
&= \frac{1}{2^\kappa} \sum_{K \in \mathbb{F}_2^\kappa} \sum_{\gamma \in \mathbb{F}_2^\kappa} \sum_{\gamma' \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma + \gamma', K \rangle} \mathrm{COR}(\alpha, \beta, \gamma) \mathrm{COR}(\alpha, \beta, \gamma') \\
&= \frac{1}{2^\kappa} \sum_{\gamma \in \mathbb{F}_2^\kappa} \sum_{\gamma' \in \mathbb{F}_2^\kappa} \mathrm{COR}(\alpha, \beta, \gamma) \mathrm{COR}(\alpha, \beta, \gamma') \underbrace{\sum_{K \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma + \gamma', K \rangle}}_{0 \text{ if } \gamma \neq \gamma', 2^\kappa \text{ if } \gamma = \gamma'} \\
&= \sum_{\gamma \in \mathbb{F}_2^\kappa} \mathrm{COR}(\alpha, \beta, \gamma)^2.
\end{aligned}$$

□

If we return to Matsui's Algorithm 2, we can show that the data complexity is dependent on $\mathrm{ELP}(\alpha,\beta)$. Indeed, following reasoning analogous to the prof of Proposition 2.6, we can show that for any fixed key the probability of success depends on the magnitude $N\mathrm{cor}_K(\alpha,\beta)^2$. The expected value of this magnitude over the keyspace is $N\mathrm{ELP}(\alpha,\beta)$, which is the reason why $\mathcal{O}\left(1/\mathrm{ELP}(\alpha,\beta)\right)$ is often used as an estimation of the data complexity of an Algorithm 2 linear attack using the approximation $\nu(\alpha,\beta)$ extended with key recovery. However, we should note that $1/\mathrm{ELP}(\alpha,\beta)$ is *not* the expected value of the data complexity over the keyspace, and just an upper bound which can be reasonably tight depending on the circumstances. A more thorough analysis of the probability of success is provided in Section 2.4.

If a single dominant mask γ exists for which $\mathrm{COR}(\alpha,\beta,\gamma)^2 \ggg \mathrm{COR}(\alpha,\beta,\gamma')^2$ for all $\gamma' \neq \gamma$, then $\mathrm{ELP}(\alpha,\beta) \simeq \mathrm{COR}(\alpha,\beta,\gamma)^2$, which means that the predictions of [Mat93] for the probability of success of Algorithms 1 and 2 were accurate. If such a dominant mask doesn't exist, then Algorithm 1 won't work as intended. However, since $\mathrm{COR}(\alpha,\beta,\gamma)^2 \leq \mathrm{ELP}(\alpha,\beta)$, we find that Proposition 2.6 often *underestimates* the probability of success of Matsui's Algorithm 2.

It should be noted that although the ELP can provide a good estimate of the usefulness of a linear approximation when there are many different contributing keyed approximations and the distribution of the correlation over the keyspace is close to a normal distribution, this is not always the case. In particular, when there are only a few approximations which have a significant contribution, the correlations follow a discrete distribution, and the ELP may prove less representative. In these cases, we often speak of "weak keys", for which the square of the correlation is much larger than the ELP might initially suggest. For the rest of the keys, the square of the correlation is much smaller than expected, or even zero.

2.2.2 Linear Hulls of Key-alternating Block Ciphers

We will now focus on the linear hulls of approximations of iterative key-alternating block ciphers (Definition 1.2). We start by momentarily ignoring the influence of the key and considering the case of a permutation of a fixed keyed instance of an iterative block cipher. Our first aim is to compute $\mathrm{cor}_K(\alpha,\beta)$ given input and output masks $\alpha,\beta \in \mathbb{F}_2^n$. A popular approach to compute the correlations of such linear approximations is that of correlation matrices, as introduced by Daemen et al. in [DGV94].

2.2.2.1 Correlation Matrices

Definition 2.19 (Correlation matrix, [DGV94]). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a Boolean function. Its correlation matrix is a $2^n \times 2^m$ matrix $C^{(f)} \in \mathrm{GL}(\mathbb{RF}_2^n, \mathbb{RF}_2^m)$ of values between -1 and 1, with*

$$C_{ij}^{(f)} = \mathrm{cor}(f;i,j) = \frac{1}{2^n}\sum_{x\in\mathbb{F}_2^n}(-1)^{\langle i,x\rangle+\langle j,f(x)\rangle}, \tag{2.30}$$

in other words, $C^{(f)}$ contains the correlations of all possible linear approximations of f.

The correlation matrix contains the correlations of all the linear approximations of the Boolean function, and has a close numerical relationship with both the LAT and the Walsh transform, which is defined in the next chapter:

$$C_{ij}^{(f)} = \frac{1}{2^{n-1}}\mathrm{LAT}_f(i,j) = \frac{1}{2^n}\widehat{f}(i,j). \tag{2.31}$$

There are also several properties which describe the correlation matrices of most components of a keyed instance of a block cipher in simple terms. These can be applied to transformations such as Sbox layers, key additions and linear layers. For example, for a function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ consisting of the xoring of a fixed constant $c \in \mathbb{F}_2^n$, $f(x) = x \oplus c$, the correlation matrix is:

$$C_{ij}^{(f)} = \begin{cases} 0 & \text{if } i \neq j \\ (-1)^{\langle i,c\rangle} & \text{if } i = j \end{cases} \quad \text{for all } i,j \in \mathbb{F}_2^n. \tag{2.32}$$

These results will be covered in detail in the context of the Walsh transform in Section 3.3 in the next chapter. All this information can be combined into a single correlation matrix thanks to the following result:

Proposition 2.20 (Correlation matrix of composition, [DGV94]). *Let $f, g : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be two vectorial Boolean functions, and let $h = g \circ f$ be their composition. The correlation matrix of the composition is the matrix product of the correlation matrices of f and g:*

$$C^{(g\circ f)} = C^{(g)} \times C^{(f)}. \tag{2.33}$$

Proof. The proof of this result is deferred to that of Proposition 3.15. □

This means that the correlations of all the linear approximations of any vectorial Boolean function consisting of the composition of several simpler maps can be obtained by multiplying the correlation matrices of each of its individual round transformations.

2.2.2.2 Linear Trails

If we apply Proposition 2.20 to a keyed instance of a key-alternating block cipher and unravel the matrix product for one of its entries, we obtain a formula for the computation of $\mathrm{cor}_K(\alpha, \beta)$:

Proposition 2.21 (Equation 31 in [DGV94]). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be an r-round key-alternating block cipher with round functions $F_i : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$, and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. Given a fixed master key K and the associated round subkeys $(K_0, \ldots, K_r)$, we have:*

$$\mathrm{cor}_K(\alpha, \beta) = \sum_{\substack{\gamma_0, \ldots, \gamma_r \in \mathbb{F}_2^n \\ \gamma_0 = \alpha, \gamma_r = \beta}} (-1)^{\langle \gamma_0, K_0 \rangle + \cdots + \langle \gamma_r, K_r \rangle} \prod_{i=1}^{r} \mathrm{cor}(F_i; \gamma_{i-1}, \gamma_i). \tag{2.34}$$

Let us examine this formula more carefully. Unlike in the piling-up lemma, we have made no assumptions on the statistical independence of any random variables, as the equation holds no matter what the round functions F_i are. It is also interesting that the formula closely resembles Equation 2.27. Indeed, we are expressing the fixed-key correlation as a sum which is parametrised by a set of key masks $\gamma = (\gamma_0, \ldots, \gamma_r)$. Each term in this sum is the product of $\prod_{i=1}^{r} \mathrm{cor}(F_i; \gamma_{i-1}, \gamma_i)$, which is independent of the key, and the sign $(-1)^{\langle \gamma_0, K_0 \rangle + \cdots + \langle \gamma_r, K_r \rangle}$, which is independent of the round functions. Each one of these sequences of intermediate masks $(\gamma_0, \ldots, \gamma_r)$, which determines a specific path of round approximations through the cipher, is called a *linear trail* of the approximation, as introduced by Daemen et al. [DGV94].

Definition 2.22 (Linear trail, [DGV94]). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be an r-round key-alternating cipher, and let $\alpha, \beta \in \mathbb{F}_2^n$. Each keyed linear approximation of the form*

$$\langle \alpha, x \rangle + \langle \gamma_0, K_0 \rangle + \cdots + \langle \gamma_r, K_r \rangle + \langle \beta, y \rangle, \tag{2.35}$$

where $\gamma_0, \ldots, \gamma_r \in \mathbb{F}_2^n$ and $\gamma_0 = \alpha,\ \gamma_r = \beta$, is called a linear trail.

We now wish to return to the case in which the key is not fixed, and provide some information on the distribution of $\mathrm{cor}_K(\alpha, \beta)$ over the keyspace. Since the key schedule may have an effect on this distribution, we will make the following assumption about the key schedule [Nyb94, DR07]:

Definition 2.23 (Long-key cipher, [DR07]). *A* long-key cipher *is an iterative key-alternating block cipher for which $\kappa = (r+1)n$ and which can be represented in such a way that the key schedule is the identity. In other words, the master key is effectively the concatenation of the round subkeys.*

In other words, in a long-key cipher all round subkeys are independent from each other and all keybits in a round subkey are independent. This assumption is false for real-use ciphers (since a long-key cipher would have an impractically long key and be vulnerable to meet-in-the-middle attacks) but it is a very useful and effective simplification when it comes to proving theoretical results. Although real-world ciphers are not long-key, in many cases we can safely assume that they behave like one. However, the key schedule does have a potentially major influence on the distribution of the correlation, as has been shown in works such as [AÅBL12]. Under the long-key cipher assumption, we note that Equations 2.34 and 2.27 become the same. From this formula and using Lemma 2.15 again we can deduce:

Proposition 2.24 (Equation 28 in [DGV94]). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^{(r+1)n} \longrightarrow \mathbb{F}_2^n$ be an r-round key-alternating long-key block cipher with round functions $F_i : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$, and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. For any linear trail $\gamma = (\gamma_0, \ldots, \gamma_r)$ we have:*

$$\mathrm{COR}(\alpha, \beta, \gamma) = \prod_{i=1}^{r} \mathrm{cor}(F_i; \gamma_{i-1}, \gamma_i). \tag{2.36}$$

From this result, we deduce the following version of the linear hull theorem for long-key ciphers, and is often assumed to hold for general key-alternating ciphers:

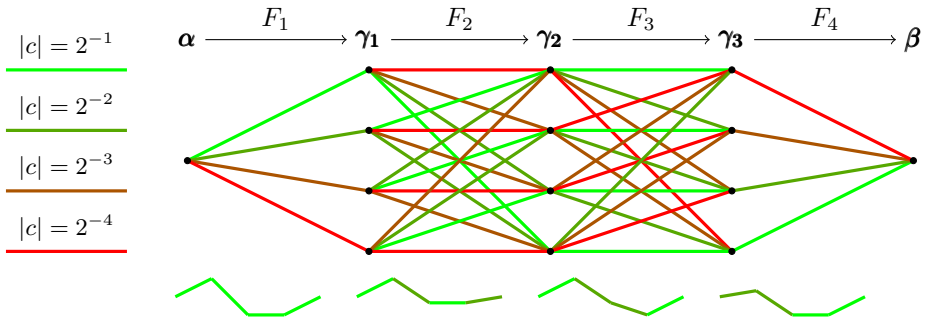


Figure 2.4: Example of the Linear Hull Theorem for a key-alternating block cipher. The trails with the largest correlations have the strongest contributions to the ELP. Although the squared correlation of the dominant trail is 2^{-8}, adding the contribution of the next three most biased trails increases the estimation of the ELP to $2^{-7.75}$. The full ELP between α and β is $2^{-7.66}$.

Corollary 2.25 (Linear hull theorem II). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be an r-round long-key cipher, and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. Then the* ELP *can be calculated as follows:*

$$\mathrm{ELP}(\alpha,\beta) = \sum_{\substack{\gamma_0,\ldots,\gamma_r \in \mathbb{F}_2^n \\ \gamma_0=\alpha,\ \gamma_r=\beta}} \prod_{i=1}^{r} \mathrm{cor}(F_i;\gamma_{i-1},\gamma_i)^2. \tag{2.37}$$

2.2.2.3 Estimating the Correlation with Sparse Correlation Matrices

Since computing the correlation for all input and output linear masks is not really possible in practice except in the case of toy examples, we often consider *truncated* correlation matrices, which were introduced by Abdelraheem [Abd12]. We consider a set of "interesting" linear masks, for example, by setting an upper bound for the Hamming weight or the number of active Sboxes. If we have already identified a set of highly-correlated linear trails, we can consider all the masks which appear in these trails. For each component of the permutation, we consider a submatrix of its correlation matrix which is obtained by selecting the rows and columns corresponding to these linear masks. By multiplying these truncated correlation matrices, we can obtain an approximation of the correlation matrix of the composition.

Another technique based on correlation matrices permits the estimation of the ELP of the linear approximations of key-alternating block ciphers. The idea is to adapt Equation 2.37 to a matrix form which can be truncated. This can be achieved by considering the component-wise square of the correlation matrices $C^{(f)}$. The product of the squared correlation matrices of the round functions of a key-alternating block cipher is composed of the ELPs of all of its linear approximations. As with traditional correlation matrices, we can consider truncated submatrices according to the number of active Sboxes or to the Hamming weight.

2.3 Multiple and Multidimensional Linear Cryptanalysis

After its initial introduction, several publications attempted to extend linear cryptanalysis in ways which would increase the effectiveness of the distinguishers. One immediately apparent way of building more useful distinguishers is to use several linear approximations. Indeed, Matsui's attacks in [Mat93, Mat94a] already make use of two approximations which recover different bits of the key so that a larger portion of the master key is recovered from the attack. However, from a statistical point of view, the approximations are used in two independent attacks, both of which must succeed on their own for the full attack to recover the correct key.

A first effort to combine several linear approximations into a single distinguisher was proposed by Kaliski and Robshaw [KR94]. This kind of attack can use several keyed linear approximations with different input and output masks, but they must all share the same key mask. This distinguisher can recover the target keybit in an Algorithm 1-type attack with a lower data complexity than would be possible using a single linear approximation. However, the requirement of a single fixed key mask is a very restrictive condition on the available linear approximations.

The first variant of linear cryptanalysis to combine information from an arbitrary set of linear approximations was introduced by Biryukov et al. [BCQ04], who provided a formal framework for statistical attacks which could be used to justify a version of Algorithm 1 using multiple keyed approximations, which is then extended to a multiple version of Algorithm 2. This technique is often called *multiple linear cryptanalysis*.

The approach of [BCQ04] still had one major limitation: it requires the assumption that the linear approximations behave as independent random variables, which is a difficult to justify assumption in many cases and demonstrably false in others. The authors argue that their predictions are still valid even if the approximations are not statistically independent, which has been found to be true in at least some experimental attacks [FN20]. However, alternative approaches which circumvent the independence assumption have also been proposed.

One way of getting around the issue is to consider the joint probability distribution of the target linear approximations as a whole. Hermelin, Cho and Nyberg introduced *Multidimensional linear cryptanalysis*, [CHN08, HCN08, HCN19], which uses a set of approximations which conform a vector subspace of the space of all possible linear approximations. This is a popular technique which can be adapted to both Algorithms 1 and 2, as well as allowing several different statistics (most notably, LLR and χ^2).

Another approach is Bogdanov et al.'s *multivariate linear cryptanalysis* [BTV18], which also avoids other assumptions, such as the normality of the correlation distribution over the keyspace, and takes the effect of the key schedule on the correlation into account. First, a large enough random sample of keys is drawn. For each key, the (signed) correlation contributions of each individual trail in a list are combined using Equation 2.34. After considering all the keys, an empirical sample of the joint distribution of the correlations of the linear approximations has been obtained. The parameters of this sample, specifically the expected values and covariances, can then be used to accurately predict the performance of multiple linear cryptanalysis statistics.

2.3.1 Using Multiple Linear Approximations

We will now give a light overview of the techniques of Biryukov et al. [BCQ04] for both Algorithms 1 and 2. We will not go into detail about the way the data complexity can be computed, as this will be covered using more recent models in Section 2.4. In this initial description, we will use counters similar to those used in the description of Matsui's Algorithms 1 and 2. They can also be described in terms of empirical correlations, which will also be discussed in Section 2.4.

2.3.1.1 Multiple Version of Algorithm 1

Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher. We consider a collection of M keyed linear approximations whose correlations are known:

$$\begin{array}{cccccccc} \nu_1: & \langle \alpha_1, x\rangle & + & \langle \beta_1, y\rangle & + & \langle \gamma_1, K\rangle, & \mathrm{COR}(\alpha_1, \beta_1, \gamma_1) = c_1, \\ \vdots & \vdots & & \vdots & & \vdots & \\ \nu_M: & \langle \alpha_M, x\rangle & + & \langle \beta_M, y\rangle & + & \langle \gamma_M, K\rangle, & \mathrm{COR}(\alpha_M, \beta_M, \gamma_M) = c_M. \end{array} \tag{2.38}$$

Given a collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N random plaintext-ciphertext pairs generated with a fixed unknown key K, we can compute the counters

$$T^i = |(x, E_K(x)) \in \mathcal{D} : \langle \alpha_i, x\rangle + \langle \beta_i, E_K(x)\rangle = 0| - \frac{N}{2}. \tag{2.39}$$

If we assume that the approximations are statistically independent, then so are the counters T^i, and their expected values are $\mathrm{Exp}_{\mathcal{D}}(T^i) = (-1)^{\langle \gamma_i, K\rangle}\frac{N}{2}c_i$. We can give a score to each guess $(z_1, \ldots, z_M)$ of the value of $(\langle \gamma_1, K\rangle, \ldots \langle \gamma_M, K\rangle)$ using the euclidean distance of $\frac{N}{2}((-1)^{z_1}c_1, \ldots, (-1)^{z_M}c_M)$ to the counter vector $(T^1, \ldots, T^M)$. In other words, we find the key guess $(z_1, \ldots, z_M) \in \mathbb{F}_2^M$ which minimises

$$T_{(z_1,\ldots,z_M)} = \sum_{i=1}^{M}\left(T^i - (-1)^{z_i}\frac{N}{2}c_i\right)^2. \tag{2.40}$$

In [BCQ04], it was shown that in order to obtain appreciable information about the key, the data complexity of this attack must be inversely proportional to the *capacity* of the set of approximations, which is defined as

$$C = \sum_{i=1}^{M}\mathrm{COR}(\alpha_i, \beta_i, \gamma_i)^2 = \sum_{i=1}^{m} c_i^2. \tag{2.41}$$

2.3.1.2 Multiple Version of Algorithm 2

Consider a block cipher which can be decomposed as $E_K = F_K \circ E'_K$, where F denotes the part covered by the key recovery, and a set of M keyless linear approximations of E' such as

$$\begin{array}{rccc} \nu_1: & \langle \alpha_1, x\rangle & + & \langle \beta_1, y\rangle, \quad \mathrm{ELP}(\alpha_1,\beta_1) = e_1, \\ \vdots & \vdots & & \vdots \\ \nu_M: & \langle \alpha_M, x\rangle & + & \langle \beta_M, y\rangle, \quad \mathrm{ELP}(\alpha_M,\beta_M) = e_M. \end{array} \tag{2.42}$$

Let f be a map so that $f(y,k)$ takes the value of $\langle \beta, F_K^{-1}(y)\rangle$ for all K which are compatible with the key guess k. Given a key guess k, we consider the counters

$$T_k^i = |(x, E_K(x)) \in \mathcal{D} : \langle \alpha_i, x\rangle + f(E_K(x), k) = 0| - \frac{N}{2}. \tag{2.43}$$

The expected value of T_k^i when the guess k is correct is $\frac{N}{2}\mathrm{cor}_K(\alpha_i,\beta_i)$. Meanwhile, for any wrong guess of the key, we can assume that the expected value is 0. Since $\mathrm{cor}_K(\alpha_i,\beta_i)$ is key-dependent, we don't know the exact distribution of the counters T_k^i for the right key. However, we can still use the following squared imbalance test statistic:

$$Q_k = \sum_{i=1}^{M} (T_k^i)^2 \tag{2.44}$$

The key guess k which maximises this statistic is likely to be the correct key. Assuming the independence of the approximations, it can be shown (see Subsection 2.4.3) that the attack is reasonably successful if the available data is around the inverse of the capacity

$$C = \sum_{i=1}^{M} \mathrm{ELP}(\alpha_i,\beta_i)^2 = \sum_{i=1}^{M} e_i. \tag{2.45}$$

2.3.2 Multidimensional Linear Cryptanalysis

Multidimensional linear cryptanalysis was introduced by Hermelin, Cho and Nyberg [HCN08, CHN08, HCN19], who proposed multidimensional versions of Matsui's Algorithms 1 and 2. Here, we will focus only on Algorithm 2, and will not go into detail about the underlying statistical reasoning. Multidimensional linear cryptanalysis uses multidimensional linear approximations, which consist of a vector subspace whose elements are traditional linear approximations:

Definition 2.26 (Multidimensional linear approximation). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher. We consider m linear approximations of E of the form $\langle \alpha_i, x\rangle + \langle \beta_i, y\rangle$ which are linearly independent, in the sense that the concatenated mask vectors $(\alpha_i|\beta_i) \in \mathbb{F}_2^{2n}$ are independent. The associated* multidimensional linear approximation *L is a vector space of dimension m whose elements are the $M = 2^m$ linear combinations of these linear approximations. There is a map $\Lambda : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ which evaluates the basis linear approximations, $\Lambda(x,y)_i = \langle \alpha_i, x\rangle + \langle \beta_i, y\rangle$. This map is given by two matrices $A, B \in \mathrm{GL}(\mathbb{F}_2^n, \mathbb{F}_2^m)$ whose rows are the α_i, β_i. The multidimensional approximation L can thus be evaluated as $\Lambda(x,y) = Ax + By$.*

For a random permutation, the distribution of the multidimensional linear approximation L is expected to be uniform, that is, $\mathrm{Exp}_P\left(\mathrm{Pr}_x(\Lambda(x,P(x)) = \eta)\right) = 1/2^m$ for all $\eta \in \mathbb{F}_2^m$. We assume that for a block cipher E we have chosen a multidimensional linear approximation whose probability distribution $p = (p_\eta)_{\eta\in\mathbb{F}_2^m}$, $p_\eta = \mathrm{Pr}_x\left(\Lambda(x, E_K(x)) = \eta\right)$ is remarkably non-uniform. The non-uniformness of this distribution can be quantified by means of the capacity of the multidimensional approximation:

Definition 2.27 (Capacity of a multidimensional approximation). *The* capacity *of a multidimensional linear approximation L for a fixed key K is the relative entropy or Kullback-Leibler distance of its probability distribution to the uniform distribution:*

$$C(K) = 2\sum_{\eta\in\mathbb{F}_2^m} p_\eta \log \frac{p_\eta}{2^{-m}} \simeq 2^m \sum_{\eta\in\mathbb{F}_2^m} (p_\eta - 2^{-m})^2 = \sum_{(\alpha|\beta)\in L\setminus\{0\}} \mathrm{cor}_K(\alpha,\beta)^2. \tag{2.46}$$

The overall capacity is the expected value of the capacity over the whole keyspace:

$$C = \mathrm{Exp}_K\left(C(K)\right) = \sum_{(\alpha|\beta)\in L\setminus\{0\}} \mathrm{ELP}(\alpha,\beta). \tag{2.47}$$

Proof. The alternative expressions of $C(K)$ are derived from Taylor expansion of the logarithm, assuming $p_\eta \simeq 2^{-m}$, and from the fact that for a fixed K we have

$$p_\eta = 2^{-m} \sum_{(\alpha|\beta)\in\mathbb{F}_2^{2n}} (-1)^{\langle(\alpha|\beta),\eta\rangle} \mathrm{cor}_K(\alpha,\beta). \qquad \square$$

In the multidimensional version of Algorithm 2, we try to distinguish the distribution p (which will describe the value of the linear approximations when the correct key guess is used) from the uniform distribution (which describes the value of the linear approximations for all other key guesses). Depending on how much information we have about the good-key distribution, we can consider two different test statistics: the log-likelihood ratio and the χ^2 test of fit statistic.

2.3.2.1 Multidimensional Linear Cryptanalysis using the LLR Statistic

When we are able to obtain a good estimation of the distribution η, we can use the log-likelihood ratio or LLR test statistic, which is known to have good properties thanks to the Neyman-Pearson lemma [NP33]. The LLR statistic is designed to distinguish between two specific known probability distributions p and q. For each partial key guess k, we are given a sample of size N of values η taken by the multidimensional linear approximation, and we want to determine which distribution it follows. We can compute an empirical approximation of this distribution:

$$\widehat{p}_\eta(k) = \frac{1}{N} \left|(x, E_K(x)) \in \mathcal{D} : Ax + BF_k^{-1}(y) = \eta\right|. \tag{2.48}$$

We can plug this empirical distribution into the formula for the LLR statistic:

Definition 2.28 (LLR statistic). *In a multidimensional linear key recovery attack for which the joint probability distribution p of the multidimensional linear approximation is known, the LLR test statistic for each key guess k can be calculated using the following formula:*

$$Q_k = \sum_{\eta\in\mathbb{F}_2^m} \widehat{p}_\eta(k) \log \frac{p_\eta}{2^{-m}}. \tag{2.49}$$

The key guess k which maximises this statistic is likely to be the correct key.

It can be shown [HCN19] that the data complexity of this attack is inversely proportional to the total capacity C of the multidimensional approximation.

2.3.2.2 Multidimensional Linear Cryptanalysis using the χ^2 Statistic

Sometimes, it may be impossible to obtain a good estimation of the probability distribution of L. This may be due to a lack of information about the multidimensional linear approximation or to the variability of this distribution over the keyspace. For this reason, it is common to use the χ^2 test-of-fit statistic, which is intended to distinguish any unknown probability distribution p from a known one q, in this case, the uniform distribution.

Definition 2.29 (χ^2 statistic). *In a multidimensional linear key recovery attack, the χ^2 test statistic for each key guess k can be calculated using the following formula:*

$$Q_k = 2^m \sum_{\eta\in\mathbb{F}_2^m} (\widehat{p}_\eta - 2^{-m})^2 = \sum_{(\alpha|\beta)\in L\setminus\{0\}} \left(\frac{1}{N} \sum_{(x,E_K(x))\in\mathcal{D}} (-1)^{\langle\alpha,x\rangle + \langle\beta, F_k^{-1}(E_K(x))\rangle} \right)^2. \tag{2.50}$$

We note that in the end we obtain a test statistic which is equivalent to the one we described for multiple linear cryptanalysis (sum of the squares of the empirical correlations). It can be shown that in this case the data complexity of the attack is inversely proportional to $C/\sqrt{2^m - 1}$, making this test statistic slightly less effective than the LLR test.

2.3.3 Conditional Linear Approximations

An alternative way of using several linear approximations which are not statistically independent is to purposefully use two or more dependent approximations in order to enhance the correlation of one of the approximations, an idea which was first introduced by Biham and Perle [BP18] and which is generally called *conditional linear cryptanalysis*.

Definition 2.30 (Conditional correlation). *Let ν be a linear approximation of a keyed instance of a block cipher E_K with input and output masks α, β. We consider another linear approximation ν_0. The correlation of ν conditioned to $\nu_0 = 0$ is defined as:*

$$\mathrm{cor}_K(\alpha, \beta | \nu_0 = 0) = \frac{1}{|x \in \mathbb{F}_2^n : \nu_0(x, E_K(x)) = 0|} \sum_{\substack{x \in \mathbb{F}_2^n \\ \nu_0 = 0}} (-1)^{\langle \alpha, x \rangle + \langle \beta, E_K(x) \rangle}. \tag{2.51}$$

In other words, the definition is the same as for the standard correlation, but we restrict the input space to the plaintexts for which the value of ν_0 is zero. The idea of conditional linear cryptanalysis is to use approximations so that the correlation of ν greatly increases if we fix the value of ν_0. We hope that the resulting improvement in data complexity due to the increased correlation overcomes the increase which is required to compensate for the discarded data. In most cases, it is sufficient that the correlation improves by a factor larger than $\sqrt{2}$:

Proposition 2.31. *We consider a classical Algorithm 2-based attack using a linear approximation ν with correlation c, which requires $N = \mathcal{O}\left(1/c^2\right)$ known plaintext-ciphertext pairs. We assume that there is a second linear approximation ν_0 so that the correlation of ν increases to $c^* > \sqrt{2}c$ if $\nu_0 = 0$. Assuming that the second linear approximation ν_0 is unbiased, an identical linear attack which rejects the plaintexts for which $\nu_0 = 1$ requires $\frac{1}{2}\left(\frac{c}{c^*}\right)^2 N < N$ known plaintext-ciphertext pairs to succeed with the same probability.*

There are several ways in which interesting conditional linear approximations can be found for different constructions. In [BP18], the linear approximation used in Matsui's attack on the DES is improved by adding a condition ν_0 which has two identical input and output masks. Due to the construction of Feistel networks, this means that ν_0 is equal to the XOR of the values of this mask at all the even rounds. By choosing this parity bit in all even rounds appropriately, it is possible to slightly enhance the correlation in each of these rounds.

Here, we will consider a slightly simpler scenario which can also be described as a conditional linear attack. Given a linear approximation of a block cipher consisting of an SPN construction (or any construction with an Sbox layer) whose correlation is mainly described by a single dominant trail, we can increase the correlation of the active Sboxes in the first and the last rounds by determining the values of some well-chosen parity bits at the Sbox input. If we fix the value of t parity bits and the correlation of the Sbox increases by a factor larger than $\sqrt{2^t}$, then the data complexity of the attack improves. Since Sboxes usually have a small number of inputs, we can find these good conditional approximations by exhaustive search.

2.4 Statistical Models for Key Recovery Attacks

In this section, we will explore in more detail the relationship between the correlation and ELP of linear approximations and the data complexity and success probability of an associated linear key recovery attack. It is intended to serve as a standalone practical guide which can be used to compute the parameters of a linear attack in most circumstances, while still providing some basic theoretical explanations. The theory behind modelling linear and other statistical attacks is a prolific topic to this day, and there is a lot of interesting bibliography. The success probability formulas found in this section mostly originate from the work of Blondeau and Nyberg [BN15, BN16, BN17], and provide a balance between accuracy and ease of use.

2.4.1 Advantage in a Statistical Key Recovery Attack

Matsui's original Algorithm 2 key recovery attack outputs a single guess k of the key, the one whose counter T_k is furthest away from $N/2$. If the attack fails and k doesn't correspond to the real key K, it is still likely that the right key will have produced one of the largest values of $|T_k - N/2|$, but the algorithm ignores this information. This problem can be fixed by modifying Algorithm 2 so that it returns a list of key guesses which surpass a certain threshold for $|T_k - N/2|$. The correct key K can thus be found with an exhaustive search step: by completing each guess k on the list to every associated master key K and performing a test encryption against a known plaintext-ciphertext pair, the correct master key can be found with high probability. This idea was already used in [Mat94a], and later formalised by Selçuk [Sel08] through the notion of advantage (not to be confused with that of distinguishing advantage):

Definition 2.32 (Key recovery advantage). *If a statistical key recovery attack algorithm involving a key guess k returns a list of $2^{|k|-a}$ key candidates, we say the attack aims for an* advantage *of a. The probability of success of the attack is the probability that the key guess corresponding to the real key belongs to the output list.*

If the algorithm provides a list of $2^{|k|-a}$ partial key candidates, then the exhaustive search requires $2^{|k|-a} \cdot 2^{\kappa-|k|} = 2^{k-a}$ test encryptions, as opposed to the standard 2^{κ} test encryptions of the generic exhaustive key search. In other words, an advantage of a means that search phase cost with respect to a brute-force attack has been reduced as if the key was a bits shorter.

An estimation of the probability of success of this type of attack was also provided in [Sel08]. Although this estimation is based on a result on order statistics, subsequent reformulations such as the one found in [BW12] have reframed the problem as a hypothesis testing scenario. Both approaches are supported by classical statistical methods which are well understood, and lead to the same results. Here we will discuss the hypothesis testing model.

As a reminder, in a statistical hypothesis test, we want to distinguish the *null hypothesis* H_0 from the *alternative hypothesis* H_1 by analysing a data sample. The effectiveness of the test is measured in two main ways. One is its *significance level* $\alpha = \Pr(H_0 \text{ rejected}|H_0)$ or probability of a false positive, which is the probability that the null hypothesis is rejected despite being true. The other is the *power* of the test $1-\beta = \Pr(H_0 \text{ rejected}|H_1)$, which is the probability that the null hypothesis is correctly rejected when the alternative hypothesis is true. For example, the distinguishing advantage of a distinguishing attack is equal to $1-\alpha-\beta$.

The statistical key recovery problem can be reformulated as a hypothesis test as follows: given a partial key guess k, we want to distinguish whether it corresponds to the actual key of the cipher K (alternative hypothesis) or is a wrong guess (null hypothesis). In this case, the power of the test $1-\beta$ is the probability that the right guess for the subkey is marked as a candidate for the search phase, in other words, the probability of success P_S of the attack. On the other hand, the significance level α is the probability that an individual wrong key guess is marked as a candidate. We thus expect $\alpha 2^{|k|}$ wrong key guesses to be marked as candidates for the search phase, and we deduce that $\alpha 2^{|k|} \simeq 2^{|k|-a}$ and we should use a test with $\alpha = 2^{-a}$ if we want an advantage of a.

We use a test statistic X_k to distinguish between both hypotheses, which is a random variable over the key K and the data $\mathcal{D}$. For example, in Matsui's Algorithm 2, this statistic is $|T_k - N/2|$. Let us suppose that, if $k = k_R$ is the right guess, then this statistic has a cumulative distribution function F_R. Analogously, we will suppose that if $k \neq k_R$ then the distribution function is F_W. We will consider a threshold Θ so that we reject the null hypothesis (that is, we keep k as a candidate) if $X_k > \Theta$ and accept it (discard the key guess) if $X_k \leq \Theta$. We have $\alpha = 1 - F_W(\Theta)$ and $P_S = 1 - \beta = 1 - F_R(\Theta)$. In practice, we choose the significance level $\alpha = 2^{-a}$ according to the desired advantage a and then take $\Theta = F_W^{-1}(1-\alpha)$. This leads to the following result, which links the advantage a and the probability of success P_S:

Theorem 2.33. *Under the previous hypothesis testing model for a statistical attack, we can bound the success probability for a given desired advantage a as follows:*

$$P_S = 1 - F_R\left(F_W^{-1}(1-2^{-a})\right), \tag{2.52}$$

where F_R and F_W denote the cumulative distribution functions of the test statistic X_k for the right and the wrong key guess, respectively. If the right-key distribution can be approximated by a normal distribution $\mathcal{N}(\mu_R, \sigma_R^2)$, we have

$$P_S \simeq \Phi\left(\frac{\mu_R - F_W^{-1}(1-2^{-a})}{\sigma_R}\right). \tag{2.53}$$

If the wrong-key distribution can also be approximated by a normal distribution $\mathcal{N}(\mu_W, \sigma_W^2)$,

$$P_S \approx \Phi\left(\frac{\mu_R - \mu_W - \sigma_W \Phi^{-1}(1-2^{-a})}{\sigma_R}\right). \tag{2.54}$$

Since $1-2^{-a}$ is usually very close to 1, using a normal approximation to compute $F_W^{-1}(1-2^{-a})$ may induce a non-negligible error because of the small slope in the tails of the normal distribution. For this reason, we recommend computing the distribution of the wrong-key statistic whenever possible, instead of taking a normal approximation.

When it comes to designing tests for linear cryptanalysis, we can follow two main approaches:

- When the distribution of the linear approximation(s) under a right key guess is known, we can use the log-likelihood ratio (LLR) statistic [BJV04], as is for example typical with multidimensional

linear cryptanalysis. According to the Neyman-Pearson lemma [NP33], for a given significance level (that is, given a desired advantage a), the LLR test has optimal power (that is, it achieves the best possible probability of success P_S).

- When the distribution of the linear approximation(s) is unknown (either due to being too difficult to compute or being key-dependent), we can still distinguish between the right key guess and the wrong guesses by using other tests: for single approximations and multiple linear cryptanalysis, we can use the difference in expected value and variance, while in multidimensional linear cryptanalysis, we use a χ^2 test-of-fit test.

2.4.1.1 Known Plaintext vs. Distinct Known Plaintext Scenarios

Traditionally, it is assumed that the plaintexts x in a known plaintext linear attack are sampled with replacement, that is, we assume that repeated plaintexts might appear. This is supported by the idea that a distinct known plaintext scenario, in which duplicate plaintexts are not allowed in the data, is in a sense "closer" to a chosen plaintext attack. However, the known plaintext with repetition scenario has its own problems: it is generally acknowledged that repeated plaintexts do not provide additional information to the attack, and in fact experimental implementations of attacks like [Mat94a, Jun01] tend to avoid the repetition of plaintexts. Furthermore, since $N = 2^n$ does not guarantee that the full codebook is available, we can consider known plaintext attacks in which the data complexity surpasses the size of the codebook: indeed, the experiments in [BN15] show that the probability of success continues to increase even after N surpasses 2^n. For these reasons, when the data complexity is close to the size of the codebook, the distinct known plaintext scenario may provide more clarity. In the remainder of this section, we will consider models which account for both scenarios.

2.4.2 Application to Matsui's Algorithm 2

We will apply this framework to Matsui's Algorithm 2. We consider an attack on the cipher $E_K = F_K \circ E'_K$ using the keyless linear approximation $\langle \alpha, x \rangle + \langle \beta, E'_K(x) \rangle$ of E'_K. If $f : \mathbb{F}_2^n \times \mathbb{F}_2^{|k|} \longrightarrow \mathbb{F}_2$ is the truncated last round function, we consider the experimental correlation:

$$\widehat{\mathrm{cor}}(k) = \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} (-1)^{\langle \alpha, x \rangle + f(y,k)} \tag{2.55}$$

The experimental correlation is equivalent to the counter $T_k - N/2$ used in the description of Algorithm 2 given in [Mat93], since $\widehat{\mathrm{cor}}(k) = 2T_k/N - 1$. Since we are interested in the key guesses with the largest absolute value of $T_k - N/2$, we will use the squared experimental correlation test statistic, as its distribution is easier to manipulate than the absolute value:

$$X_k = \left(\widehat{\mathrm{cor}(k)}\right)^2 \tag{2.56}$$

In the following subsections, we will compute the distribution of this statistic both for the right key and the wrong key guess, so that we can plug them into Theorem 2.33.

2.4.2.1 Distribution of the Correlation of a Linear Approximation over the Keyspace

Let us start by substituting the right-key and wrong-key randomisation hypotheses for updated versions of these assumptions as formulated by Blondeau and Nyberg [BN16].

Assumption 2.34 (Right-key randomisation). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. If K is uniformly distributed over $\mathbb{F}_2^\kappa$, then*

$$\mathrm{cor}_K(\alpha, \beta) \sim \mathcal{N}(c, \mathrm{ELP}(\alpha, \beta) - c^2). \tag{2.57}$$

In general $c = \mathrm{Exp}_K(\mathrm{cor}_K(\alpha, \beta))$ is close to or equal to zero.

Let us provide some justification to this assumption. The value of the variance is deduced directly from the definition of the ELP as the expected value over the keyspace of the square of the correlation. The assumption that $c = 0$ is a consequence of the design of modern ciphers, which aim to avoid the existence of linear approximations with any dominant linear trails. Furthermore, even for a single trail, we can expect that as many keys will contribute to the average correlation with a positive sign as will contribute with a negative sign, thus leading to an expected value of zero.

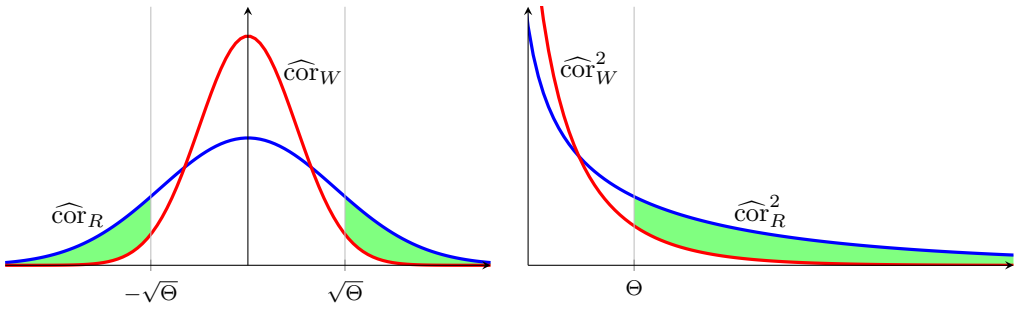


Figure 2.5: Probability density functions of the statistical distributions of the equivalent linear cryptanalysis test statistics $\widehat{\mathrm{cor}}(k)$ and $\widehat{\mathrm{cor}}(k)^2$ for the right and the wrong key guess. The area highlighted in green represents the distinguishing advantage of the test statistic.

The normality of the distribution is more difficult to justify. When a single dominant linear trail γ is present, we can separate the keyspace into two parts $\mathcal{K}_0$ and $\mathcal{K}_1$ depending on the value of $\langle K, \gamma \rangle$. The expected values of $\mathrm{cor}_K(\alpha, \beta)$ in both groups take opposite sign, and their absolute value is very close to the square root of the ELP. This means that the variance within each group is very close to zero and that $\mathrm{cor}_K(\alpha, \beta)$ only takes two values and is not normally distributed. However, in most modern cipher constructions there are no large sets of dominant trails in any linear approximations after enough rounds, and we expect that the combination of the contributions of each trail will lead to a normal distribution. Several experiments [DR07, BW12] have shown that this is a reasonable assumption for most modern block ciphers, although exceptions exist, see for example [AÅBL12, HV18].

We also require an effective way of estimating the ELP of a linear approximation. Since it is generally unfeasible to account for all the trails in the linear hull, we can only identify a set of dominant trails which have a large contribution to the linear potential. In other words, we consider a set $\mathcal{S}$ of key masks γ so that the approximations $\langle \alpha, x \rangle + \langle \beta, y \rangle + \langle \gamma, K \rangle$ are highly biased, while the rest have a much smaller correlation. The contribution of these less biased approximations is modelled as random noise, an idea introduced in [BN16].

Theorem 2.35 (Theorem 3 in [BN16]). *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \longrightarrow \mathbb{F}_2^n$ be a block cipher and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks. We can estimate the* ELP *by choosing a set $\mathcal{S}$ containing the linear trails contributing the most to the correlation, and using the formula:*

$$\mathrm{ELP}(\alpha, \beta) \simeq \sum_{\gamma \in \mathcal{S}} \mathrm{COR}(\alpha, \beta, \gamma)^2 + 2^{-n}. \tag{2.58}$$

For the wrong-key distribution of the correlation, we will assume that the linear approximation behaves as a linear approximation of a random permutation.

Theorem 2.36 (Theorem 4.7 in [DR07]). *Let $P : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a permutation chosen uniformly at random and let $\alpha, \beta \in \mathbb{F}_2^n$ be input and output masks.*

$$\mathrm{cor}(P; \alpha, \beta) \sim \mathcal{N}(0, 2^{-n}). \tag{2.59}$$

Proof. A thorough proof of this result is given in [DR07]. □

Assumption 2.37 (Wrong-key randomisation). *For a wrong key guess k, the random variable over K*

$$\frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle + f(y,k)},$$

behaves as the correlation of a a linear approximation of a random permutation and thus follows the normal distribution $\mathcal{N}(0, 2^{-n})$. In addition, we will assume that the correlations for each different wrong key guess are statistically independent from each other.

2.4.2.2 Distribution of the Right and Wrong-key Statistics

We can now compute the distribution of $\widehat{\mathrm{cor}}(k)$ in both the right and the wrong key case. We can see both as random variables over the data sample and the key. In order to lighten notation, we will denote by $\widehat{\mathrm{cor}}_R$ and $\widehat{\mathrm{cor}}_W$ two independent random variables which have identical distributions to $\widehat{\mathrm{cor}}(k)$ for the right key guess and for any wrong key guess, respectively.

In order to consider both the known plaintext and the distinct known plaintext scenarios, a constant B representing the current case will be used:

$$B = \begin{cases} 1 & \text{in the KP scenario} \\ \frac{2^n - N}{2^n - 1} & \text{in the DKP scenario} \end{cases}. \tag{2.60}$$

Proposition 2.38. *Given a linear approximation with masks $\alpha, \beta \in \mathbb{F}_2^n$ and a random data sample $\mathcal{D}$, the experimental correlation has the following expected value and variance:*

$$\mathrm{Exp}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) = \mathrm{Exp}_K(\mathrm{cor}_K(\alpha,\beta)) = c, \tag{2.61}$$

$$\mathrm{Var}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) = \frac{B}{N} + \mathrm{ELP}(\alpha,\beta) - c^2. \tag{2.62}$$

Assuming there is a large enough number of dominant linear trails for the approximation, $\widehat{\mathrm{cor}}_R$ follows a normal distribution. In most cases, $\mathrm{Exp}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) = 0$.

Proof. We initially consider that the key K is fixed. This means that $\langle \alpha, x\rangle + \langle \beta, E'_K(x)\rangle + 1$ is drawn from the Bernoulli distribution with parameter $(1 + \mathrm{cor}_K(\alpha,\beta))/2$. Therefore

$$|(x, y = E'_K(x)) \in \mathcal{D} : \langle \alpha, x\rangle + \langle \beta, y\rangle = 0| \sim \begin{cases} \mathcal{B}\left(\frac{1+\mathrm{cor}_K(\alpha,\beta)}{2}, N\right) & \text{for KP} \\ \mathcal{HG}\left(\frac{1+\mathrm{cor}_K(\alpha,\beta)}{2}2^n, 2^n, N\right) & \text{for DKP} \end{cases}.$$

From this we deduce that, for a fixed key, we can approximate $\widehat{\mathrm{cor}}_R \sim \mathcal{N}(\mathrm{cor}_K(\alpha,\beta), \frac{B}{N})$. If we now consider a uniformly distributed key, we have the following:

$$\begin{aligned} \mathrm{Exp}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) &= \mathrm{Exp}_K\left(\mathrm{Exp}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) = \mathrm{Exp}_K(\mathrm{cor}_K(\alpha,\beta)), \\ \mathrm{Var}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) &= \mathrm{Exp}_K\left(\mathrm{Var}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) + \mathrm{Var}_K\left(\mathrm{Exp}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) \\ &= \frac{B}{N} + \mathrm{ELP}(\alpha,\beta) - \mathrm{Exp}_K(\mathrm{cor}(\alpha,\beta))^2. \end{aligned}$$

□

Proposition 2.39. *The wrong-key experimental correlation follows the distribution:*

$$\widehat{\mathrm{cor}}_W \sim \mathcal{N}\left(0, \frac{B}{N} + 2^{-n}\right). \tag{2.63}$$

Proof. We reason in the same way as for the right key. $\langle \alpha, x\rangle + f(y,k) + 1$ is drawn from the Bernoulli distribution with parameter $(1 + c(K))/2$, where $c(K) \sim_K \mathcal{N}(0, 2^{-n})$. Therefore

$$|(x, y = E'_K(x)) \in \mathcal{D} : \langle \alpha, x\rangle + f(y,k) = 0| \sim \begin{cases} \mathcal{B}\left(\frac{1+c(K)}{2}, N\right) & \text{for KP} \\ \mathcal{HG}\left(\frac{1+c(K)}{2}2^n, 2^n, N\right) & \text{for DKP} \end{cases}.$$

From this we deduce that, for a fixed key, $\widehat{\mathrm{cor}}_W \sim \mathcal{N}(c(K), \frac{B}{N})$. If we now consider a uniformly distributed key, we have the following:

$$\begin{aligned} \mathrm{Exp}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) &= \mathrm{Exp}_K\left(\mathrm{Exp}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) = 0, \\ \mathrm{Var}_{\mathcal{D},K}(\widehat{\mathrm{cor}}_R) &= \mathrm{Exp}_K\left(\mathrm{Var}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) + \mathrm{Var}_K\left(\mathrm{Exp}_{\mathcal{D}}(\widehat{\mathrm{cor}}_R)\right) \\ &= \frac{B}{N} + 2^{-n}. \end{aligned}$$

□

Using these normal approximations for the right and wrong-key experimental correlations, we can model their squares as χ^2 distributions with 1 degree of freedom:

Corollary 2.40. *If we assume that $c = \mathrm{Exp}\left(\widehat{\mathrm{cor}}_K(\alpha,\beta)\right) = 0$, we can model the distributions of the square experimental correlations as χ^2 distributions with one degree of freedom:*

$$\frac{N}{B + N\mathrm{ELP}(\alpha,\beta)}\widehat{\mathrm{cor}}_R^2 \sim \chi_1^2, \tag{2.64}$$

$$\frac{N}{B + N2^{-n}}\widehat{\mathrm{cor}}_W^2 \sim \chi_1^2. \tag{2.65}$$

The expected values and variances of these distributions are:

$$\mu_R = \frac{B}{N} + \mathrm{ELP}(\alpha,\beta), \qquad \sigma_R^2 = 2\left(\frac{B}{N} + \mathrm{ELP}(\alpha,\beta)\right)^2, \tag{2.66}$$

$$\mu_W = \frac{B}{N} + 2^{-n}, \qquad \sigma_W^2 = 2\left(\frac{B}{N} + 2^{-n}\right)^2. \tag{2.67}$$

These distributions can be plugged in into the formulas of Theorem 2.33. In general, we will obtain more accurate results if we do not take the normal approximation of the distribution of the wrong-key statistic. However, the one-sided test on the square experimental correlation is equivalent to a two-sided test on the experimental correlation itself (see Figure 2.5), which we can use to deduce the following result:

Corollary 2.41 (Theorem 2 in [BN16]). *In an Algorithm 2 linear attack aiming for an advantage a and using an approximation with expected linear potential $\mathrm{ELP}(\alpha, \beta)$ with the squared experimental correlation test statistic, the probability of success is:*

$$P_S = 2 - 2\Phi\left(\sqrt{\frac{B + N2^{-n}}{B + N\mathrm{ELP}(\alpha,\beta)}} \cdot \Phi^{-1}(1 - 2^{-a-1})\right). \tag{2.68}$$

2.4.3 Application to Multiple and Multidimensional Linear Cryptanalysis

After analysing the classical Algorithm 2 attack using a single linear approximation, we now proceed to extend the results to multiple and multidimensional attacks.

2.4.3.1 On the Independence Assumption

When discussing multiple linear cryptanalysis, we already mentioned that [BCQ04] requires the assumption that all the linear approximations used in the attack are independent from each other, and that approaches such as multidimensional [HCN19] and multivariate [BTV18] tackle this issue by considering the joint distribution of the linear approximations.

However, these approaches can have some disadvantages: multidimensional cryptanalysis requires that we incorporate a lot of additional approximations which may have poor correlations, and thus only contribute to the test statistic with additional noise. In the case of multivariate linear cryptanalysis, it is not possible to compute the data complexity and success probability directly from the parameters of the approximation, as it is necessary to perform the empirical estimation of the joint distribution of the correlations. Given these observations, studying the effectiveness of "classical" multiple linear cryptanalysis using the χ^2 statistic on an arbitrary set of approximations is a problem which may still prove worthwhile. Indeed, some multiple linear attacks such as [FN20] have shown experimental results which closely follow the predictions of the model, despite the clear falseness of the independence assumption (in the case of these attacks, the approximations are not only statistically but linearly dependent).

One possible approach would be to see whether weaker conditions than statistical independence hold. In particular, pairwise statistical independence or even statistical incorrelation should be enough to control both the expected value and the variance of the χ^2 statistic. This would require a better understanding of the statistical dependence of linear approximations. In [Nyb17], it is proven that in a vector space of binary random variables which are pairwise statistically independent, linear independence is equivalent to statistical independence. However, it was also shown that any such space of pairwise independent binary random variables is essentially degenerate, in the sense that all variables must be either constant or balanced, which makes the result of little use to linear cryptanalysis. Furthermore, simple examples are also shown in which two linear approximations must necessarily be statistically dependent, for example a pair of linear approximations which share the same input mask but have different output masks.

For these reasons, it seems highly unlikely that these attacks are working as predicted by the model because of the approximations being pairwise independent. In that case, there are several possible explanations. For example, it is possible that the contribution of the covariance between approximations is small enough compared to the correlation of each approximation for its contribution to the variance of the χ^2 statistic to be negligible. It is also possible that there is some other underlying theoretical reason. For example, it may be possible that the multiple linear attacks for which the model succeeds can be described as multidimensional attacks from which low correlation approximations have been removed. In any case, it would be of great interest to explain this phenomenon so that we have a better understanding of the kind of scenarios the classical models for which multiple linear cryptanalysis are effective.

2.4.3.2 Distribution of the Capacity

In order to estimate the success probability in the case of multiple and multidimensional linear cryptanalysis, we start by finding the distribution of the capacity of the set of approximations:

$$C(K) = \sum_{i=1}^{M} \mathrm{cor}_K(\alpha_i, \beta_i)^2. \tag{2.69}$$

Theorem 2.42 (Theorems 4,5 in [BN16]). *Let us consider a set of M linear approximations $\nu_i : \langle \alpha_i, x\rangle + \langle \beta_i, E_{K'}(x)\rangle$, of which we know representative families $\mathcal{S}_i$ of linear trails for the first M_0. The expected value and variance of the capacity can be estimated as*

$$Exp_K\left(C(K)\right) = \sum_{i=1}^{M} ELP(\alpha_i, \beta_i) \simeq \sum_{i=1}^{M_0} \sum_{\gamma \in \mathcal{S}_i} \mathrm{COR}(\alpha_i, \beta_i, \gamma)^2 + M2^{-n}, \tag{2.70}$$

$$Var_K(C(K)) = 2\sum_{i=1}^{M_0} \left(\sum_{\gamma \in \mathcal{S}_i} \mathrm{COR}(\alpha_i, \beta_i, \gamma)^2 \right)^2 + 2^{2-n} \sum_{i=1}^{M_0} \sum_{\gamma \in \mathcal{S}_i} \mathrm{COR}(\alpha_i, \beta_i, \gamma)^2 + M2^{1-2n}. \tag{2.71}$$

If only an estimate C of the capacity of the first M_0 linear approximations of the set is known, we can still approximate the variance by

$$Var_K(C(K)) \simeq \frac{2}{M_0} C^2 + \frac{M - M_0}{M_0} C 2^{2-n} + \frac{M - M_0}{M_0} 2^{1-2n}. \tag{2.72}$$

Proof. These formulas are deduced from the linearity of the expected value and the linearity of the variance for independent random variables. We use our previous estimation for the linear potential in Theorem 2.35, and assume all linear approximations for which no trails are known behave like noise. For the formula in the case in which only a capacity estimate is known, we assume that the M_0 approximations have the same correlation. □

2.4.3.3 Distribution of the Right-key and Wrong-key Statistics

The final component of the model is the computation of the distribution of the right and wrong key statistics over K and $\mathcal{D}$, which can then be plugged into the formulas of Theorem 2.33. We again consider both the KP and the DKP scenarios, which will determine the value of the parameter B, which was defined in equation 2.60. We consider the statistic

$$Q_k = \sum_{i=1}^{M} \widehat{\mathrm{cor}}_i(k), \text{ where } \widehat{\mathrm{cor}}_i(k) \text{ is the empirical correlation of } \nu_i. \tag{2.73}$$

We will denote by Q_R and Q_W random variables which have the same distribution as Q_k in the right-key and the wrong-key scenarios, respectively. From the previous results, we deduce the following theorem:

Theorem 2.43 (Theorem 6 in [BN16]). *In multiple and multidimensional linear cryptanalysis, the statistic Q_R approximately follows a normal distribution with parameters*

$$Q_R \sim \mathcal{N}(\mu_R, \sigma_R^2), \text{ where } \left\{ \begin{array}{rcl} \mu_R & = & \frac{B}{N} M + \mathrm{Exp}_K\left(C(K)\right) \\ \sigma_R^2 & = & 2\frac{B^2}{N^2} M + 4\frac{B}{N} \mathrm{Exp}_K\left(C(K)\right) + \mathrm{Var}_K\left(C(K)\right) \end{array} \right. . \tag{2.74}$$

A multiple of the wrong key statistic follows a χ^2 distribution with M degrees of freedom:

$$\frac{N}{B + N2^{-n}} Q_W \sim \chi^2_M, \text{ so } \left\{ \begin{array}{rcccl} \mu_W & = & \mathrm{Exp}_{\mathcal{D},K}(Q_{\tilde{k}}) & = & M\left(\frac{B}{N} + 2^{-n}\right) \\ \sigma_W^2 & = & \mathrm{Var}_{\mathcal{D},K}(Q_{\tilde{k}}) & = & 2M(\frac{B}{N} + 2^{-n})^2 \end{array} \right. . \tag{2.75}$$

Proof. We will begin by deducing the distribution of the right-key statistic. From the proof of Proposition 2.38, we know that, for a fixed value of the key K, we have $\widehat{\mathrm{cor}}_K(\alpha_i, \beta_i) \sim \mathcal{N}(\mathrm{cor}_K(\alpha_i, \beta_i), \frac{B}{N})$. Assuming the independence of the approximations, for a fixed K we have:

$$\frac{N}{B} Q_R \sim \chi^2_M \left(\frac{N}{B} C(K) \right), \text{ so } \left\{ \begin{array}{rcl} \mathrm{Exp}_{\mathcal{D}}\left(Q_R\right) & = & \frac{B}{N} M + C(K) \\ \mathrm{Var}_{\mathcal{D}}\left(Q_R\right) & = & 2\frac{B^2}{N^2} M + 4\frac{B}{N} C(K) \end{array} \right. .$$

Therefore, assuming a normal approximation of the χ^2 distribution,

$$\begin{aligned} \mathrm{Exp}_{\mathcal{D},K}\left(Q_R\right) &= \mathrm{Exp}_K\left(\mathrm{Exp}_{\mathcal{D}}\left(Q_R\right)\right) = \frac{B}{N} M + \mathrm{Exp}_K\left(C(K)\right), \\ Var_{\mathcal{D},K}\left(Q_R\right) &= \mathrm{Exp}_K\left(\mathrm{Var}_{\mathcal{D}}\left(Q_R\right)\right) + \mathrm{Var}_K\left(\mathrm{Exp}_{\mathcal{D}}\left(Q_R\right)\right) \\ &= 2\frac{B^2}{N^2} M + 4\frac{B}{N} \mathrm{Exp}_K\left(C(K)\right) + \mathrm{Var}_K\left(C(K)\right). \end{aligned}$$

We now proceed to the wrong-key statistic. From Proposition 2.39, we know that for each approximation $\widehat{\mathrm{cor}}_i(\tilde{k}) \sim \mathcal{N}\left(0, \frac{B}{N} + 2^{-n}\right)$ when K and $\mathcal{D}$ are uniformly distributed. Since all these variables have the same variance, we can normalise them so that the addition of their squares follows a χ^2 distribution with M degrees of freedom. □

2.5 Differential-linear Cryptanalysis

In this section we will discuss a related family of attacks which was introduced by Langford and Hellman [LH94] and combines a differential and a linear approximation into a single distinguisher. In [LH94], only differential characteristics of probability 1 were considered, so here we will describe the slightly improved version introduced by Biham et al. [BDK02a], which allows for differentials with probability other than 1.

Proposition 2.44 (Differential-linear distinguisher). *Let $E_K : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a block cipher which can be written as the composition of two separate parts E_K^1 and E_K^2, that is, $E_K = E_K^2 \circ E_K^1$. We consider that a differential of E_K^1 and a linear approximation of E_K^2 are known. We denote the differential of E_K^1 by $\Delta \longrightarrow \Delta^*$, and assume it has probability p:*

$$\mathrm{DP}(\Delta \xmapsto{E_K^1} \Delta^*) = p. \tag{2.76}$$

Let $\langle \alpha, y\rangle + \langle \beta, z\rangle$ be a linear approximation of E_K^2 with correlation q:

$$\mathrm{cor}\left(E_K^2; \alpha, \beta\right) = q. \tag{2.77}$$

The differential and the linear approximation can be combined into an expression of the form $\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle$, which showcases the following correlation:

$$\Pr\left(\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle = 0\right) - \Pr\left(\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle = 1\right) = \pm pq^2, \tag{2.78}$$

under the appropriate independence assumptions. This magnitude is closely-related to the concept of the autocorrelation of a Boolean function, and we will thus denote it by $\mathrm{aut}(E_K; \Delta, \beta)$.

Proof. If we assume that $\langle \alpha, y\rangle + \langle \beta, E_K^2(y)\rangle$ and $\langle \alpha, y+\Delta^*\rangle + \langle \beta, E_K^2(y+\Delta^*)\rangle$ are independent binary variables which are zero with probability $\frac{1+q}{2}$ when y is uniformly distributed, using the piling-up lemma we can deduce that the expression

$$\langle \alpha, y\rangle + \langle \alpha, y+\Delta^*\rangle + \langle \beta, E_K^2(y)\rangle + \langle \beta, E_K^2(y+\Delta^*)\rangle = \langle \alpha, \Delta^*\rangle + \langle \beta, E_K^2(y)\rangle + \langle \beta, E_K^2(y+\Delta^*)\rangle$$

is zero with probability $\frac{1+q^2}{2}$. This means that $\mathrm{aut}(E_K^2; \Delta^*, \beta) = \pm q^2$ depending on the value of $\langle \alpha, \Delta^*\rangle$. Assuming that the correlation of the linear approximation of E_K^2 is independent of whether the differential holds or not, using the Bayes formula we obtain:

$$\begin{aligned}
&\Pr\left(\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle = 0\right) \\
&\quad = \Pr\left(E_K^1(x+\Delta) = E_K^1(x) + \Delta^*\right) \cdot \Pr\left(\frac{\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle = 0}{E_K^1(x+\Delta) = E_K^1(x) + \Delta^*}\right) \\
&\quad + \Pr\left(E_K^1(x+\Delta) \neq E_K^1(x) + \Delta^*\right) \cdot \Pr\left(\frac{\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x+\Delta)\rangle = 0}{E_K^1(x+\Delta) \neq E_K^1(x) + \Delta^*}\right) \\
&\quad = p\left(\frac{1 \pm q^2}{2}\right) + (1-p)\cdot\frac{1}{2} = \frac{1 \pm pq^2}{2}.
\end{aligned}$$

From this, we quickly deduce that $\mathrm{aut}(E_K; \Delta, \beta) = \pm pq^2$. □

In practice, this means that E_K can be distinguished from a random permutation with $O\left(p^2q^4\right)$ chosen plaintext-ciphertext pairs. It suffices to generate plaintext pairs $(x, x \oplus \Delta)$ and compute the correlation of $\langle \beta, E_K(x)\rangle + \langle \beta, E_K(x \oplus \Delta)\rangle$. Differential-linear cryptanalysis is of particular interest in the case of block ciphers in which the optimal differential probability and the optimal correlation decrease very rapidly with the number of rounds.

A differential-linear distinguisher can be used to mount a key-recovery attack by extending it at the top and/or at the bottom with key guesses. The key recovery in the initial rounds is handled in the same way as in differential attacks using plaintext structures. The key recovery in the final rounds is handled using Matsui's Algorithm 2. By computing the correlation of the differential-linear distinguisher for each combined guess of both parts of the key, we can separate the correct key guess from all the wrong ones. The data complexity and the probability of success can be determined using the same techniques used in linear cryptanalysis, such as the ones presented in Section 2.4.

In practice, this description of differential-linear distinguishers can lead to rather inaccurate results which do not match those obtained in experiments. The presence of truncated differentials and linear hulls will influence the correlation of a differential-linear distinguisher. It is also possible that there are

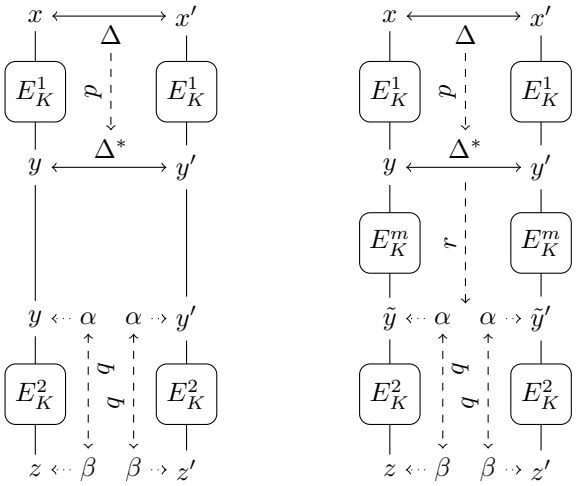


Figure 2.6: On the left, the "classical" differential-linear distinguisher without a transition part, which has correlation pq^2. On the right, the differential-linear distinguisher which uses a transition step between the differential and the linear approximation, and which has correlation prq^2. The correlation of the middle part E_K^m can be evaluated either empirically or using special techniques, such as the DLCT.

several output differences Δ^* and input masks α which contribute to the autocorrelation. Furthermore, the independence assumptions used in the proof of the result are not usually satisfied, so the transition between the differential and the linear part may have an unexpected effect on the autocorrelation.

The most commonly used solution to these problems consists of separating E_K into three parts, $E_K = E_K^2 \circ E_K^m \circ E_K^1$, where E_K^m is a small transition layer which covers only one or two rounds. E_K^1 and E_K^2 are still covered using a differential and a linear approximation, respectively. If $r = \mathrm{aut}(E_K^m; \Delta^*, \alpha)$ is the autocorrelation for the transition layer, then it can be easily proven that $\mathrm{aut}(E_K; \alpha, \beta) = prq^2$ (see Figure 2.6). This approach describes the behaviour at the transition between the differential and the linear parts much more accurately than the previous model.

There are two approaches which can be used to compute the value of $\mathrm{aut}(E_K^m; \Delta^*, \alpha)$. The first is simply through experimental simulation, as since E_K^m only covers one or two rounds, the autocorrelation should be large enough to be accurately estimated with a relatively small amount of randomly-generated plaintext pairs. It is also possible to use theoretical tools which aim to describe the behaviour of the transition, most notably the differential-linear connectivity table or DLCT [BDKW19]. These approaches have the advantage that they also allow us to search for differences Δ^* and linear masks α which exhibit particularly high autocorrelations.

Another improvement to differential-linear cryptanalysis is the idea introduced in [BLT20], which consists of grouping plaintext pairs in such a way that all the pairs in a group either do or don't satisfy the differential at the same time. For example, if we find some bits in the plaintext for which flipping their value won't change whether the differential is satisfied or not, we can use them to determine such groups. The autocorrelation in a "good" group (i.e. the differential is satisfied) will be rq^2, and zero in the "bad" groups. If each group has more than $1/r^2q^4$ pairs, then we can perform Algorithm 2 on each group until we find one for which correlation is observed: this requires $\mathcal{O}\left(pr^2q^4\right)$ plaintext-ciphertext pairs instead of $\mathcal{O}\left(p^2r^2q^4\right)$. This technique can still be used if the "good" groups are not entirely made of good pairs. If, given a differential pair, the probability that another pair in the same group is also a differential pair is $p' > p$, then the data complexity is $\mathcal{O}\left(pp'r^2q^4\right)$.

Chapter 3

Boolean Functions and the Walsh Transform

This chapter contains broad introductions to some mathematical concepts which appear frequently in this thesis, in particular, the Walsh transform of both complex vectors and Boolean functions. The former is of interest to us mainly because of the fast Walsh transform algorithm, which can be used to greatly reduce the time complexity of most linear key recovery attacks on block ciphers, as will be shown in Chapter 5. The latter is a topic which is of interest in many branches of symmetric cryptology.

Contents

Section 3.1 focuses entirely on the multidimensional discrete Fourier transform on complex vectors. We start from an algebraic perspective looking at the discrete Fourier transform on finite Abelian groups and then deduce a more traditional formulation. Section 3.2 features an in-depth explanation of the (complex) fast Walsh transform algorithm. Finally, Section 3.3 explores the Walsh transform of vectorial Boolean functions, its properties, and its relationship to linear cryptanalysis.

3.1 (Complex) Discrete Fourier Transforms

Over the course of this section, we will define the multidimensional discrete Fourier transform for vectors in $\mathbb{C}^N$, of which the Walsh transform is a specific case. We start by describing the Fourier transform on finite Abelian groups in order to highlight the importance of the algebraic structure in regard to the fast transform algorithms.

3.1.1 Fourier Transform on Finite Abelian Groups

We will now provide an extremely succinct description of the Fourier transform on finite Abelian groups. It is not strictly necessary for the purposes of this text, but I believe some basic context can help illustrate the importance of the underlying group structure of the discrete Fourier transform, as in particular it explains some of the differences in the behaviour of the one-dimensional FFT and the fast Walsh transform when it comes to pruning. Please note that it is not intended as a real introduction to the topic and all concepts and results which are not essential to our purposes have been omitted, as well as any detailed proofs. For a more detailed study of Fourier analysis on finite groups, we refer the reader to [Ter99].

Definition 3.1 (Group characteristics). *Let G be a finite Abelian group. A* characteristic *of G is any group homomorphism $\chi : G \longrightarrow \mathbb{C}^\times$, where $\mathbb{C}^\times$ is the multiplicative group of the complex numbers. The characteristics of G form themselves a group $\widehat{G} = \mathrm{Hom}(G, \mathbb{C}^\times)$ called the* Pontryagin dual *of G. In the case of finite Abelian groups, the dual $\widehat{G}$ is isomorphic to G.*

It can be shown that all characteristics χ of a group G must map all elements of G to the unit circle $S^1 = \{z \in \mathbb{C} : |z| = 1\}$. Please note that although the groups G and $\widehat{G}$ are isomorphic, there is usually no unique or canonical isomorphism between the two.

Example. Let $G = \mathbb{Z}/N\mathbb{Z}$ be the cyclic group of N elements, and let $\chi : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{C}$ be a characteristic. Since we must have $\chi(1)^N = 1$, we deduce that $\chi(1)$ must be an N-th root of unity. Let ζ be an N-th primitive root of 1. Then $\chi(1) = \zeta^k$ for some k between 0 and $N-1$. Furthermore, we can construct the following group isomorphism ϕ:

$$\begin{array}{rccc} \phi: & \dfrac{\mathbb{Z}}{N\mathbb{Z}} & \xrightarrow{\quad\cong\quad} & \widehat{\dfrac{\mathbb{Z}}{N\mathbb{Z}}} \\ & k & \longmapsto & \chi_k, \quad \chi_k(l) = \left(\zeta^k\right)^l = \zeta^{k\cdot l} \end{array}.$$

Since there are multiple different choices for ζ, the isomorphism ϕ is not uniquely determined.

Example. Let $G = \mathbb{F}_2^n$ with the additive group structure, and let $\chi : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a characteristic. Since all elements of $\mathbb{F}_2^n$ have order 2 except for the vector 0, we deduce that $\chi(x) = \pm 1$ for all x, which means that we can think of χ as a linear map between $\mathbb{F}_2^n$ and $\mathbb{F}_2$. Since all linear maps are of the form $\langle \gamma, \cdot \rangle$ for some $\gamma \in \mathbb{F}_2^n$, we obtain an isomorphism

$$\begin{array}{rccc} \phi: & \mathbb{F}_2^n & \xrightarrow{\quad\cong\quad} & \widehat{\mathbb{F}_2^n} \\ & \gamma & \longmapsto & \chi_\gamma, \quad \chi_\gamma(x) = (-1)^{\langle \gamma, x \rangle} \end{array}.$$

For every bijective linear map $L \in \mathrm{GL}(\mathbb{F}_2^n, \mathbb{F}_2^n)$, the bilinear form $\langle \gamma, Lx \rangle$ leads to a similar isomorphism between G and $\widehat{G}$. In other words, we cannot choose a group isomorphism without choosing a basis of $\mathbb{F}_2^n$.

Definition 3.2 (Space of complex functions). *Let G be a finite Abelian group. We denote the $\mathbb{C}$-vector space of all functions $f : G \longrightarrow \mathbb{C}$ using the standard addition and pointwise multiplication by $\mathbb{C}G$. Since an arbitrary map f is determined by the images of each element of G, we can see that $\mathbb{C}G$ is isomorphic to $\mathbb{C}^{|G|}$. We can thus carry over the inner product and norm:*

$$\langle f, g \rangle = \sum_{x \in G} \overline{f(x)} g(x), \quad f, g \in \mathbb{C}G, \tag{3.1}$$

$$\|f\|_2 = \sqrt{\langle f, f \rangle} = \sqrt{\sum_{x \in G} |f(x)|^2}, \quad f \in \mathbb{C}G. \tag{3.2}$$

We note that, even though $\mathbb{C}G$ can be identified as a vector space with $\mathbb{C}^{|G|}$, in most cases we can find several different Abelian groups with the same order. Broadly speaking, given $\mathbb{C}^N$, choosing a group G with $|G| = N$ imposes the group structure of G on the coordinates of $\mathbb{C}^N$ in the standard basis. Different group structures will lead to different Fourier transforms on $\mathbb{C}^N$.

Definition 3.3 (Fourier transformation). *Let $f \in \mathbb{C}G$ be a function $f : G \longrightarrow \mathbb{C}$. The Fourier transform of f is a function $\widehat{f} \in \mathbb{C}\widehat{G}$, $\widehat{f} : \widehat{G} \longrightarrow \mathbb{C}$, which is defined as*

$$\widehat{f}(\chi) = \sum_{x \in G} \overline{\chi(x)} f(x), \quad \chi \in \widehat{G}. \tag{3.3}$$

The map $\mathcal{F} : \mathbb{C}G \longrightarrow \mathbb{C}\widehat{G}$ assigning $\mathcal{F}(f) = \widehat{f}$ is called the Fourier transformation.

The following result can be obtained rather quickly from the definition, however, it has wide-ranging implications, and several properties about the DFT can be deduced from it directly.

Proposition 3.4. *The Fourier transformation $\mathcal{F}$ is a linear bijection which preserves the inner product up to a constant, as $\langle \widehat{f}, \widehat{g} \rangle = |G| \langle f, g \rangle$ for all $f, g \in \mathbb{C}G$. This means that $\sqrt{|G|}\|\widehat{f}\|_2 = \|f\|_2$, so $\mathcal{F}/\sqrt{|G|}$ is a unitary operation. In addition, the inverse of $\mathcal{F}$ is $\mathcal{F}^{-1} = \mathcal{F}^*/|G|$, where $\mathcal{F}^*$ denotes the conjugate transpose operator.*

3.1.2 Multidimensional Discrete Fourier Transform

After this broad algebraic view of the Fourier transform, let us now look at it from a lower perspective. We start with the examples of the DFT and the Walsh transform, and finally show how we have in fact defined the multidimensional DFT.

Example (Discrete Fourier transform). Let $G = \mathbb{Z}/N\mathbb{Z}$. A vector $f \in \mathbb{C}G$ can also be understood as a vector in $\mathbb{C}^N$ with coordinates $(f(0), f(1), \ldots, f(N-1))$ in the standard basis. Using the identification of $\widehat{\mathbb{Z}/N\mathbb{Z}}$ with $\mathbb{Z}/N\mathbb{Z}$ we discussed earlier, we can see $\mathcal{F}$ as a map which takes vectors in $\mathbb{C}^N$ to vectors in $\mathbb{C}^N$. If we pick the primitive root of unity $\zeta = e^{2\pi i/N}$, we have:

$$\widehat{f}(k) = \sum_{x=0}^{N-1} \overline{\chi_k(x)} f(x) = \sum_{x=0}^{N-1} \overline{\zeta^{k\cdot x}} f(x) = \sum_{x=0}^{N-1} e^{-i\frac{2\pi}{N}kx} f(x), \tag{3.4}$$

which is indeed the well-known formula for the (one-dimensional) discrete Fourier transform $\mathcal{F}$ on a complex vector of length N.

Example (Walsh transform). Let $G = \mathbb{F}_2^n$. A vector in $\mathbb{C}G$ can also be seen as a vector in $\mathbb{C}^{2^n}$ whose coordinates in the standard basis are indexed by elements of $\mathbb{F}_2^n$, or elements of $\mathbb{Z}/2^n\mathbb{Z}$ if we take their binary representation. If we consider the isomorphism between $\mathbb{F}_2^n$ and its Pontryagin dual given by the standard inner product $\langle \cdot, \cdot \rangle$, we obtain the expression

$$\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} \overline{\chi_u(x)} f(x) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x \rangle} f(x), \tag{3.5}$$

which is the classical formula for the Walsh transform $\mathcal{W} : \mathbb{C}^{2^n} \longrightarrow \mathbb{C}^{2^n}$.

Since any finite Abelian group is isomorphic to a direct product of d cyclic groups of the form

$$G = \frac{\mathbb{Z}}{N_1\mathbb{Z}} \times \frac{\mathbb{Z}}{N_2\mathbb{Z}} \times \cdots \times \frac{\mathbb{Z}}{N_d\mathbb{Z}},$$

we can study all Fourier transforms on finite Abelian groups by studying the ones associated to this family of groups. In fact, we obtain all the multidimensional DFTs:

Definition 3.5 (Multidimensional discrete Fourier transform). *Let $d \in \mathbb{N}$ and $N_1, \ldots, N_d \in \mathbb{N}$, $N = N_1 \cdot \cdots \cdot N_d$. Given a map $f : \frac{\mathbb{Z}}{N_1\mathbb{Z}} \times \cdots \times \frac{\mathbb{Z}}{N_d\mathbb{Z}} \longrightarrow \mathbb{C}$, we define the $(N_1, \ldots, N_d)-$dimensional discrete Fourier transform of f as the map $\widehat{f} : \frac{\mathbb{Z}}{N_1\mathbb{Z}} \times \cdots \times \frac{\mathbb{Z}}{N_d\mathbb{Z}} \longrightarrow \mathbb{C}$ given as*

$$\widehat{f}(k_1, \ldots, k_d) = \sum_{x_1=0}^{N_1-1} \cdots \sum_{x_d=0}^{N_d-1} e^{-i\frac{2\pi}{N_1}k_1x_1 - \cdots - i\frac{2\pi}{N_d}k_dx_d} f(x_1, \ldots, x_d). \tag{3.6}$$

We can see f and $\widehat{f}$ as elements of $\mathbb{C}^N$, and the Fourier transform as a map $\mathcal{F} : \mathbb{C}^N \longrightarrow \mathbb{C}^N$. The input space is often called the time domain *and the output space is often called the* frequency domain. *The multidimensional discrete Fourier transform can be inverted using the formula*

$$f(x_1, \ldots, x_d) = \frac{1}{N} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} e^{i\frac{2\pi}{N_1}k_1x_1 + \cdots + i\frac{2\pi}{N_d}k_dx_d} \widehat{f}(k_1, \ldots, k_d). \tag{3.7}$$

Of special interest are the cases:

- *The case $d = 1$ corresponds to the discrete Fourier transform of length $N = N_1$.*
- *The case $d = n$, $N_1 = \cdots = N_d = 2$ corresponds to the Walsh transform of length $N = 2^n$.*

The following result contains some interesting properties of the discrete Fourier transform, most of which can be derived directly from Proposition 3.4:

Theorem 3.6. *The multidimensional discrete Fourier transformation operator $\mathcal{F} : \mathbb{C}^N \longrightarrow \mathbb{C}^N$ given by $\mathcal{F}(f) = \widehat{f}$ satisfies the following properties:*

- **Linearity:** *$\mathcal{F}$ is a linear operation. Given $f, g \in \mathbb{C}^N$ and $a, b \in \mathbb{C}$, we have*

$$\mathcal{F}(af + bg) = a\mathcal{F}(f) + b\mathcal{F}(g). \tag{3.8}$$

- **Shift theorem:** *Given fixed time shifts $s_i \in \mathbb{Z}/N_i\mathbb{Z}, \quad i = 1, \ldots, d$ and a map $f \in \mathbb{C}^N$, we define the shifted map f_{+s} as $f_{+s}(x_1, \ldots, x_d) = f(x_1 - s_1, \ldots, x_d - s_d)$. The DFT exchanges shifts with fixed frequency phase factors, according to the formula:*

$$\widehat{f_{+s}}(k_1, \ldots, k_d) = e^{-i\frac{2\pi}{N_1}k_1s_1 - \cdots - i\frac{2\pi}{N_d}k_ds_d} \widehat{f}(k_1, \ldots, k_d). \tag{3.9}$$

- **Parseval's identity:** *$\mathcal{F}$ preserves the dot product up to a factor of N. Given $f, g \in \mathbb{C}^N$:*

$$\langle \widehat{f}, \widehat{g} \rangle = N \langle f, g \rangle, \qquad \|\widehat{f}\|_2 = \sqrt{N} \|f\|_2. \tag{3.10}$$

These results are often written as the so-called Parseval and Plancherel identities:

$$\sum_{x_1=0}^{N_1-1} \cdots \sum_{x_d=0}^{N_d-1} \overline{f(x_1, \ldots, x_d)} g(x_1, \ldots, x_d) = \frac{1}{N} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} \overline{\widehat{f}(k_1, \ldots, k_d)} \widehat{g}(k_1, \ldots, k_d), \tag{3.11}$$

$$\sum_{x_1=0}^{N_1-1} \cdots \sum_{x_d=0}^{N_d-1} |f(x_1, \ldots, x_d)|^2 = \frac{1}{N} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} |\widehat{f}(k_1, \ldots, k_d)|^2. \tag{3.12}$$

A consequence of Parseval's identity is the fact that $\mathcal{F}$ preserves orthogonality. Furthermore, given an orthonormal basis of $\mathbb{C}^N$, we can obtain another orthonormal basis by applying $\mathcal{F}$ and dividing by $1/\sqrt{N}$.

- **Convolution theorem:** *Given two functions $f, g \in \mathbb{C}^N$, we define their convolution as another function $f * g \in \mathbb{C}^N$ which is given by*

$$(f * g)(x_1, \ldots, x_d) = \sum_{t_1=0}^{N_1-1} \cdots \sum_{t_d=0}^{N_d-1} f(t_1, \ldots, t_d) g(x_1 - t_1, \ldots, x_d - t_d). \tag{3.13}$$

The convolution theorem states that the DFT of the convolution of two functions is the component-wise product of their DFTs:

$$\widehat{f * g}(k_1, \ldots, k_d) = \widehat{f}(k_1, \ldots, k_d) \cdot \widehat{g}(k_1, \ldots, k_d). \tag{3.14}$$

This formula can be used to compute the convolution of functions efficiently.

- **Maximum value:** *The Plancherel identity tells us that $\mathcal{F}$ behaves very well with respect to the 2-norm $\|\cdot\|_2$. We are also interested in its behaviour with respect to the ∞-norm $\|\cdot\|_\infty$, about which we can say $\|\mathcal{F}\|_\infty = N$, which in particular means*

$$|\widehat{f}(k_1, \ldots, k_d)| \leq N \cdot \max \{ |f(x_1, \ldots, x_d)| : x_1 \in \mathbb{Z}/N_1\mathbb{Z}, \ldots, x_d \in \mathbb{Z}/N_d\mathbb{Z} \} \tag{3.15}$$

for all $k_1 \in \mathbb{Z}/N_1\mathbb{Z}, \ldots, k_d \in \mathbb{Z}/N_d\mathbb{Z}$.

Proof. Most of these properties are direct consequences of Proposition 3.4 or can be proven with a small manipulation of the definition. We will provide a short direct proof of the convolution theorem nonetheless, due to its importance in later chapters.

$$\begin{aligned}
\widehat{f * g}(k_1, \ldots, k_d) &= \sum_{x_1, \ldots, x_d} e^{-\sum_j i \frac{2\pi}{N_1} k_j x_j} \sum_{t_1, \ldots, t_d} f(t_1, \ldots, t_d) g(x_1 - t_1, \ldots, x_d - t_d) \\
&= \left(\sum_{t_1, \ldots, t_d} e^{-\sum_j i \frac{2\pi}{N_1} k_j t_j} f(t_1, \ldots, t_d) \right) \left(\sum_{x_1, \ldots, x_d} e^{-\sum_j i \frac{2\pi}{N_1} k_j (x_j - t_j)} g(x_1 - t_1, \ldots, x_d - t_d) \right) \\
&= \widehat{f}(k_1, \ldots, k_d) \cdot \widehat{g}(k_1, \ldots, k_d).
\end{aligned}$$

□

Other properties and applications of the (one-dimensional) DFT can be found in signal processing textbooks such as [PM92]. A coverage oriented towards the multidimensional DFT can be found in [DM90].

3.2 The Fast Walsh Transform Algorithm

The fast Fourier transform algorithm or FFT was introduced in Cooley and Tukey's seminal paper [CT65], and, thanks to more practically-oriented descriptions such as [GS66] and the wide range of potential application problems in which it can be of use, it quickly became a ubiquitous algorithm. Using a divide-and-conquer approach, the time complexity of obtaining the DFT of a vector of length N is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. Although it was originally introduced for the purposes of the one-dimensional DFT, it can easily be extended to the multidimensional case, and in particular to the Walsh transform. In this section, we will focus on the fast Walsh transform algorithm, which will be described from two different perspectives.

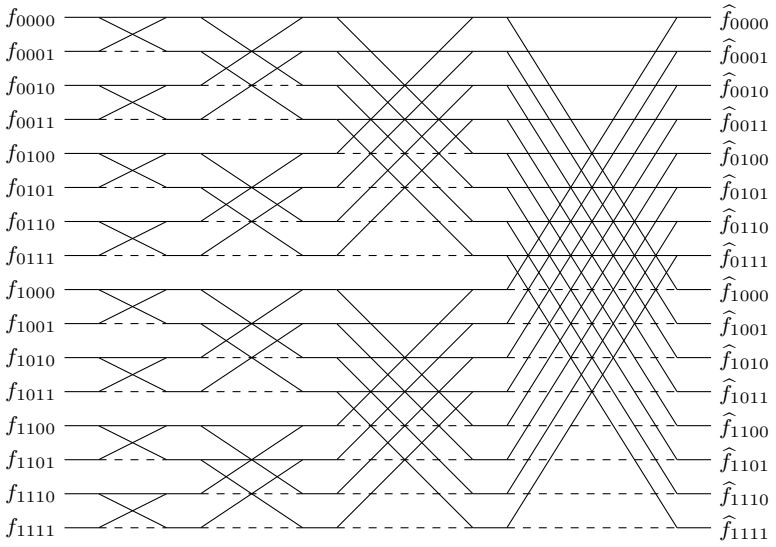


Figure 3.1: The fast Walsh transform algorithm, starting from the least significant bit, and going to the most significant ($\pi = \mathrm{id}$). The dashed lines denote a sign swap or subtraction.

3.2.1 Divide-and-conquer Description

The core idea of the fast Walsh transform algorithm is to reduce the computation of the transform of length 2^n to two smaller transforms of sizes 2^{n-1}, and repeat this operation recursively until the resulting transforms are trivial. Let $f \in \mathbb{C}\mathbb{F}_2^n$ be the vector whose Walsh transform we wish to compute. The coordinates of $\widehat{f}$ are

$$\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x \rangle} f(x). \tag{3.16}$$

Let $x = (x_{n-1}, \ldots, x_0)$ and $u = (u_{n-1}, \ldots, u_0)$ be the binary expressions of x and u, respectively. We can expand the formula of the definition as follows:

$$\widehat{f}(u_{n-1}, \ldots, u_0) = \sum_{x_{n-1} \in \mathbb{F}_2} \cdots \sum_{x_0 \in \mathbb{F}_2} (-1)^{u_{n-1}x_{n-1} + \cdots + u_0 x_0} f(x_{n-1}, \ldots, x_0). \tag{3.17}$$

Let us focus on any of the bit positions in the output index, for example u_{n-1}. We can separate the formula into two parts according to the value of x_{n-1}:

$$\begin{aligned} \widehat{f}(u_{n-1}, \ldots, u_0) = & \sum_{x_{n-2} \in \mathbb{F}_2} \cdots \sum_{x_0 \in \mathbb{F}_2} (-1)^{u_{n-2}x_{n-2} + \cdots + u_0 x_0} f(0, x_{l-2}, \ldots, x_0) \\ & + (-1)^{u_{n-1}} \sum_{x_{n-2} \in \mathbb{F}_2} \cdots \sum_{x_0 \in \mathbb{F}_2} (-1)^{u_{n-2}x_{n-2} + \cdots + u_0 x_0} f(1, x_{l-2}, \ldots, x_0). \end{aligned} \tag{3.18}$$

By inspecting this formula, we see that we have reduced the Walsh transform to two of half the size. Indeed, let's consider the functions $f_0, f_1 \in \mathbb{C}\mathbb{F}_2^{n-1}$ which are defined as $f_0(x_{n-2}, \ldots, x_0) = f(0, x_{n-2}, \ldots, x_0)$ and $f_1(x_{n-2}, \ldots, x_0) = f(1, x_{n-2}, \ldots, x_0)$ and assume we have already computed their Walsh transforms. We can obtain $\widehat{f}$ by adding $\widehat{f}_0$ and $\widehat{f}_1$ component by component (thus obtaining all the coordinates of $\widehat{f}$ for which $u_{n-1} = 0$) and by subtracting $\widehat{f}_1$ from $\widehat{f}_0$ (thus obtaining all the coordinates of $\widehat{f}$ for which $u_{n-1} = 1$).

This process can be iterated: we can reduce the computation of the two transforms of length 2^{n-1} to that of four transforms of length 2^{n-2}, and so on. Each time we reduce a transform of size 2^l to two transforms of size 2^{l-1}, we have to perform 2^l overhead additions and subtractions. These add up to 2^n additions and subtractions for each level of the reduction or *stage*. We note that at each stage, we operate on pairs (a, b) by applying the operation $(a, b) \leftarrow (a + b, a - b)$. These operations are often called *butterflies* in the literature. By the end of the process and after $n2^n$ additions and subtractions, we have reduced the calculation to that of 2^n Walsh transforms of length 1, which are trivial. The whole process is illustrated for a transform of length $2^4 = 16$ in Figure 3.1. The following result follows from this reasoning:

Theorem 3.7 (Fast Walsh transform). *Let $f \in \mathbb{C}\mathbb{F}_2^n$ be a complex vector of length 2^n. Its Walsh transform $\widehat{f}$ can be computed using exactly $n2^n$ complex additions and subtractions.*

Algorithm 7: Fast Walsh transform, "binary" description

Parameters: $\pi \in S_n$ a permutation of the set $\{1, \ldots, n\}$.
Input: $f : \mathbb{C}\mathbb{F}_2^n \longrightarrow \mathbb{C}$.
Output: $\widehat{f} : \mathbb{C}\mathbb{F}_2^n \longrightarrow \mathbb{C}$.
for $k \leftarrow \pi(1)$ **to** $\pi(n)$ **do** // The FWT "stages"
  foreach $i \in \mathbb{F}_2^n$ *with* $i \wedge (1 \ll (k-1)) = 0$ **do** // The "butterflies"
    $j \leftarrow i \oplus (1 \ll (k-1))$;
    $(f(i), f(j)) \leftarrow (f(i) + f(j), f(i) - f(j))$;
  end
end
return f;

Algorithm 8: Fast Walsh transform, "arithmetic" description

Parameters: $\pi \in S_n$ a permutation of the set $\{1, \ldots, n\}$.
Input: $f : \mathbb{C}^{2^n} \longrightarrow \mathbb{C}$.
Output: $\widehat{f} : \mathbb{C}^{2^n} \longrightarrow \mathbb{C}$.
for $k \leftarrow \pi(1)$ **to** $\pi(n)$ **do** // The FWT "stages"
  for $i \leftarrow 0$ **to** $2^n - 1$ **do** // The "butterflies"
    if $i \bmod 2^k \in \{0, \ldots, 2^{k-1} - 1\}$ **then**
      $j \leftarrow i + 2^{k-1}$;
      $(f(i), f(j)) \leftarrow (f(i) + f(j), f(i) - f(j))$;
    end
  end
end
return f;

Let us briefly consider the memory requirements of the algorithm we have outlined. Assuming that the whole input vector f is stored in memory as either one or two arrays of 2^n real values (depending on whether the input is real or complex), we can perform all the operations of the algorithm directly on this array. The only overhead is a single auxiliary register which is used in the computation of the butterflies, as well as the counters for the loops.

Another important feature of this algorithm is the multiple ways in which it can be ordered due to the commutativity of its steps. If we return to Equation 3.18, we see that our choice of wedge variable is not limited to u_{n-1}, and that we can instead start from u_0 or any other position. Furthermore, in each successive reduction step, we can choose to continue with any of the remaining variables, or even continue with different variables on each of the two smaller Walsh transforms. In order to keep the notation light, we will only formalise the former case. In short, we can consider a family of fast Walsh transform algorithms of length 2^n which is parametrised by the symmetric group S_n.

Given a permutation π of $1, \ldots, n$, the associated fast Walsh transform algorithm starts by reducing the problem on the variable $u_{\pi(n)-1}$, continues with $u_{\pi(n-1)-1}$, and so on, until only $u_{\pi(1)-1}$ remains. Since the reduction takes place at the end of the algorithm, this means the algorithm will start with the stage pairing vector entries with different values of $u_{\pi(1)-1}$ and end with the stage pairing vector entries with different values of $u_{\pi(n)-1}$.

Algorithms 7 and 8 are two pseudocode implementations of the fast Walsh transform algorithm after unrolling the recurrence. Algorithm 7 has the input and output vectors indexed in $\mathbb{F}_2^n$, while Algorithm 8 sees the vectors as indexed in $\mathbb{Z}/2^n\mathbb{Z}$. When implemented on a real computer using binary integers, both descriptions become almost indistinguishable, but both have been included in order to help the reader understand the algorithm in depth.

Both versions of the algorithm take a permutation $\pi \in S_n$ as a parameter which determines the order of the stages. An index variable k is used to keep track of the stage of the transformation, and it takes the values $\pi(1), \pi(2), \ldots, \pi(n)$. A second index variable i is then used to go over all the elements of $\mathbb{F}_2^n$ whose $(k-1)$-th coordinate j_{k-1} is equal to 0. A butterfly operation is then performed on the entries of the vector f_i and $f_{i \oplus e_{k-1}}$, where e_{k-1} is the vector composed of all zeroes except for the $(k-1)$-th coordinate.

In practice, instead of considering all possible values of $i \in \mathbb{F}_2^n$ and discarding those for which $i_{k-1} = 1$, it is possible to generate only the interesting values by using two nested for loops:

```
foreach r ∈ F_2^{n-k} do
    foreach s ∈ F_2^{k-1} do
        i ← s ⊕ (r ≪ k);
    end
end
```

```
for r ← 0 to 2^{n-k} − 1 do
    for s ← 0 to 2^{k-1} − 1 do
        i ← s + r · 2^k;
    end
end
```

In Figure 3.1, we can see that the butterflies in stage k can be grouped into 2^{n-k} groups consisting of 2^{k-1} each (for example, in stage 2 we have four groups of two butterflies). In the two nested for loops, the variable r represents the group, while the variable s selects the individual butterfly within the group.

3.2.2 Hadamard-Sylvester Matrix Description

Let us now briefly describe the fast Walsh transform in terms of Walsh-Hadamard matrices. These matrices and the several different row and column reorderings constitute a family of great mathematical interest in their own right. A lot of results about these matrices can be found in the literature, such as in [YH97].

Definition 3.8 (Hadamard-Sylvester matrices). *The matrices H_{2^n}, where*

$$\begin{aligned} H_1 &= (1) && \in \mathrm{GL}(\mathbb{C},\mathbb{C}),\\ H_2 &= \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} && \in \mathrm{GL}(\mathbb{CF}_2,\mathbb{CF}_2),\\ &\vdots \\ H_{2^n} &= H_2 \otimes H_{2^{n-1}} && \in \mathrm{GL}(\mathbb{CF}_2^n,\mathbb{CF}_2^n), \end{aligned} \tag{3.19}$$

are called Hadamard-Sylvester matrices.

In some literature, the matrix we denote by H_{2^n} is denoted simply by H_n. This recursive definition can be unrolled into a compact formula, as per the following result:

Proposition 3.9. *The entries of the Hadamard-Sylvester matrix H_{2^n} are precisely:*

$$H_{2^n}[i,j] = (-1)^{\langle i,j\rangle} \text{ for all } i,j \in \mathbb{F}_2^n. \tag{3.20}$$

Proof. We proceed by induction on n. We can easily see that the result is true for H_1 and H_2 by checking coordinate by coordinate. Furthermore, for the matrix H_{2^n}, we have

$$H_{2^n}[i,j] = (H_2 \otimes H_{2^{n-1}})[i,j] = H_2[i_{n-1},j_{n-1}] \cdot H_{2^{n-1}}[(i_{n-2},\ldots,i_0),(j_{n-2},\ldots,j_0)].$$

Assuming that the target equality holds for $H_{2^{n-1}}$, we obtain

$$H_{2^n}[i,j] = (-1)^{i_{n-1}\cdot j_{n-1}} \cdot (-1)^{\langle (i_{n-2},\ldots,i_0),(j_{n-2},\ldots,j_0)\rangle} = (-1)^{\langle i,j\rangle}. \qquad \square$$

From this result, it becomes clear that H_{2^n} is the matrix representation of the Walsh transform operator $\mathcal{W}$, in the sense that given $f \in \mathbb{CF}_2^n$, then the equality $\widehat{f} = H_{2^n} f$ holds. In fact, several texts use the Hadamard-Sylvester matrices as the definition of the Walsh transform.

Using the Hadamard-Sylvester matrices, the fast Walsh transform algorithm can be deduced in an alternative way by means of the following result:

Proposition 3.10. *Given any permutation $\pi \in S_n$ of the set $\{1,\ldots,n\}$, the Hadamard-Sylvester matrix H_{2^n} can be decomposed as*

$$H_{2^n} = \prod_{k=1}^{n} \left(I_{2^{n-\pi(k)}} \otimes H_2 \otimes I_{2^{\pi(k)-1}}\right). \tag{3.21}$$

In particular, for the case $\pi = \mathrm{id}$, we obtain the decomposition

$$H_{2^n} = \prod_{k=1}^{n} \left(I_{2^{n-k}} \otimes H_2 \otimes I_{2^{k-1}}\right). \tag{3.22}$$

Proof. We use the mixed product property of the matrix and Kronecker products:

$$H_{2^n} = \bigotimes_{k=1}^{n} H_2 = \bigotimes_{k=1}^{n} \left(I_2^{n-\pi(k)} H_2 I_2^{\pi(k)-1}\right) = \prod_{k=1}^{n} \left(I_{2^{n-\pi(k)}} \otimes H_2 \otimes I_{2^{\pi(k)-1}}\right),$$

noting that $\bigotimes_{k=1}^{l} I_2 = I_{2^l}$. $\square$

We note that all these decompositions describe H_{2^n} as the product of n matrices. For all the decompositions, the same matrices appear, only in a different order which is determined by π. For the sake of convenience, we will rename the matrices as $H^k_{2^n}$:

$$H^k_{2^n} = I_{2^{n-k}} \otimes H_2 \otimes I_{2^{k-1}}. \tag{3.23}$$

For example, the following are the three matrices for $n = 3$:

$$H^1_{2^3} = I_2 \otimes I_2 \otimes H_2 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix},$$

$$H^2_{2^3} = I_2 \otimes H_2 \otimes I_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix},$$

$$H^3_{2^3} = H_2 \otimes I_2 \otimes I_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

We can see that in order to compute the Walsh transform of a given vector of length 2^n, we just have to multiply it by all the matrices $H^k_{2^n}$, in any order. So, in order to provide a Walsh transform algorithm, we just need to find how to multiply a vector by the matrix $H^k_{2^n}$ efficiently. The entries of $H^k_{2^n}$ are generated using the Kronecker product formula

$$H^k_{2^n}[i,j] = I_{2^{n-k}}[(i_{n-1},\ldots,i_{k+1}),(j_{n-1},\ldots,j_{k+1})] \cdot H_2[i_k,j_k] \cdot I_{2^{k-1}}[(i_{k-1},\ldots,i_1),(j_{k-1},\ldots,j_1)], \tag{3.24}$$

from which we deduce that there are only two non-zero elements in each row and column:

$$H^k_{2^n}[i,j] \neq 0 \Longleftrightarrow (i_{n-1},\ldots,i_{k+1}) = (j_{n-1},\ldots,j_{k+1}) \text{ and } (i_{k-1},\ldots,i_1),(j_{k-1},\ldots,j_1). \tag{3.25}$$

Furthermore, in that case we have

$$H^k_{2^n}[(i_{n-1},\ldots,i_{k+1},i_k,i_{k-1},\ldots,i_0),(i_{n-1},\ldots,i_{k+1},j_k,i_{k-1},\ldots,i_0)] = (-1)^{i_k \cdot j_k}. \tag{3.26}$$

This suggests multiplication of $f \in \mathbb{F}_2^n$ by $H^k_{2^n}$ can be performed with the following loop:

foreach $i \in \mathbb{F}_2^n$ *with* $i_k = 0$ **do**
  $j \leftarrow i \oplus (1 \ll k)$;
  $(f(i), f(j)) \leftarrow (f(i) + f(j), f(i) - f(j))$;
end

This means multiplication by $H^k_{2^n}$ is performed in exactly the same way as the k-th stage of Algorithm 7, and we have obtained the same fast Walsh transform algorithm that we described in the previous section.

3.2.3 Comparison with the One-dimensional FFT

We finish the discussion of the fast Walsh transform algorithm by describing the classical radix-2 one-dimensional FFT algorithm and highlighting some major differences. There are fundamentally two versions of the Cooley-Tukey radix-2 FFT algorithm for vectors of length 2^n: decimation in time and decimation in frequency. We will discuss decimation in time, as the decimation in frequency algorithm can be obtained by applying decimation in time to the inverse transform.

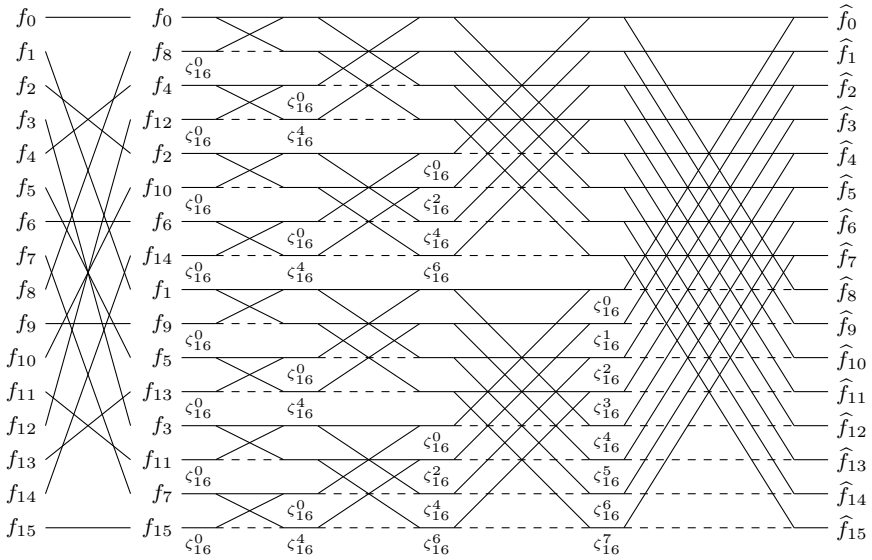


Figure 3.2: The decimation-in-time FFT algorithm.

Let $\zeta_N = e^{-i\frac{2\pi}{N}}$, and let f be a complex vector of length $N = 2^n$. The idea of the decimation-in-time FFT is to split the transform into two smaller transforms: one on the even coordinates of f and one on the odd coordinates of f.

$$\begin{aligned}\widehat{f}(k) &= \sum_{x=0}^{2^n-1} \zeta_{2^n}^{k\cdot x} f(x) \\ &= \sum_{y=0}^{2^{n-1}-1} \zeta_{2^n}^{k\cdot 2y} f(2y) + \sum_{y=0}^{2^{n-1}-1} \zeta_{2^n}^{k\cdot(2y+1)} f(2y+1) \\ &= \underbrace{\sum_{y=0}^{2^{n-1}-1} \zeta_{2^{n-1}}^{(k \bmod 2^{n-1})\cdot y} f(2y)}_{\widehat{f_{\text{even}}}(k \bmod 2^{n-1})} + \zeta_{2^n}^{k} \underbrace{\sum_{y=0}^{2^{n-1}-1} \zeta_{2^{n-1}}^{(k \bmod 2^{n-1})\cdot y} f(2y+1)}_{\widehat{f_{\text{odd}}}(k \bmod 2^{n-1})}.\end{aligned} \tag{3.27}$$

This expression leads to a reduction of the transform of length 2^n to two transforms of length 2^{n-1}, f_{even} and f_{odd}. This is repeated recursively to obtain the decimation-in-time FFT algorithm. In practice, it is common to make use of the fact that $\zeta_{2^n}^{2^{n-1}+k} = -\zeta_{2^n}^k$ by multiplying $\widehat{f_{\text{odd}}}(k), 0 \le k < 2^{n-1}$ by the factors $\zeta_{2^n}^k$ and inverting the sign when $k \ge 2^{n-1}$ is needed. A full diagram of this algorithm is found in Figure 3.2. There are three important differences with the fast Walsh transform:

- The algorithm described above features, in addition to sign swaps, the so-called *twiddle factors*, which are the multiplications by $\zeta_{2^n}^k$. This greatly complicates the application of the ideas we will use for the pruned fast Walsh transform.
- The FFT algorithm features a bit-reversal step (at the beginning for decimation-in-time and at the end for decimation-in-frequency) which is not present in the FWT.
- There are fundamentally only two radix-2 versions of the FFT algorithm, as opposed to the $n!$ which are possible for the FWT. Higher radix FFTs also exist, but they are very different, as the transform is divided into more than two reduced transforms. The larger number of transform algorithm means the FWT allows greater flexibility when pruning.

3.3 Walsh Transforms of Vectorial Boolean Functions

Since most symmetric cryptography manipulates objects belonging to vector spaces over either $\mathbb{F}_2$ or other finite fields of characteristic 2, the analysis and classification of vectorial Boolean functions can provide a lot of tools which can be of interest to cryptologists. Indeed, there are many properties of Boolean functions which must be taken into account when designing symmetric primitives, as is proven by the

existence of compendiums such as [Car21]. In particular, this section focuses on the Walsh transform, which can be used to describe the properties of different constructions relating to linear cryptanalysis. The results shown in this section became known to cryptologists thanks to the work on correlation matrices of Daemen et al. [DGV94].

Definition 3.11 (Walsh transform of a vectorial Boolean function). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be any vectorial Boolean function. We define its Walsh transform $\widehat{f}$ as the complex matrix corresponding to the map $\widehat{f} : \mathbb{F}_2^n \times \mathbb{F}_2^m \longrightarrow \mathbb{C}$ given by the formula*

$$\widehat{f}(u,v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x\rangle + \langle v,f(x)\rangle}. \tag{3.28}$$

In the case $m = 1$ we do not need the output mask v as $\widehat{f}(u,0)$ is trivial, which means we can simply consider a vector $\widehat{f} : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ given by the formula

$$\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x\rangle + f(x)}. \tag{3.29}$$

The Walsh transform of a vectorial Boolean function is a real-valued complex matrix whose columns are the transforms of each of the linear components of the function. Each column $\widehat{f}(\cdot, v)$ is the (complex) Walsh transform of the indicator map

$$\begin{array}{rccc} \mathrm{ind}_{f,v} : & \mathbb{F}_2^n & \longrightarrow & \mathbb{C} \\ & x & \longmapsto & (-1)^{\langle v,f(x)\rangle} \end{array}. \tag{3.30}$$

Given the Walsh transform of a vectorial Boolean function, the function can be recovered by applying the inverse Walsh transform on each of the columns of the matrix.

We note that the Walsh transform of f is the correlation matrix $C^{(f)}$ multiplied by a constant factor 2^n. This means that we can use the fast Walsh transform to compute the LAT of Sboxes efficiently, as the cost of computing each column of the LAT of an n-bit to m-bit Sbox is reduced from the 2^{2n} operations required if we use the definition to just $n2^n$ operations. We can thus compute the LAT of such an Sbox with $n2^{n+m}$ operations. The fast Walsh transform can also be used to find high-correlation linear approximations empirically:

Example. Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a vectorial Boolean function. We assume that we wish to find a linear approximation with a high correlation of the form $\langle \alpha, x\rangle + \langle \beta, f(x)\rangle$, where β is known but α is unknown. If we wish to find linear approximations with a correlation larger than c_0, we consider a sample of $N = \mathcal{O}\left(1/c_0^2\right)$ values of x and their images $f(x)$. We construct a zero vector g of length 2^n, and for each x in the random sample $\mathcal{D}$ we make $g(x) = (-1)^{\langle \beta, f(x)\rangle}$. The components of the Walsh transform of this vector are $\widehat{g}(\alpha) = \sum_{x \in \mathcal{D}} (-1)^{\langle \alpha,x\rangle + \langle \beta,f(x)\rangle}$, which is N times the empirical correlation of the linear approximation $\langle \alpha, x\rangle + \langle \beta, f(x)\rangle$. This means we have found the correlations for all linear approximations with input mask α with $N + n2^n$ operations instead of the $N2^n$ required operations if we just follow the definition.

A lot of the properties of the Walsh transform of complex-valued functions on $\mathbb{F}_2^n$ (sometimes called *pseudoboolean functions*) translate to properties of the Walsh transform of vectorial Boolean functions. Of particular use is the following extended version of Lemma 2.15.

Lemma 3.12 (Equation 4 in [DGV94]). *Let $u, v \in \mathbb{F}_2^n$. Then*

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x\rangle + \langle v,x\rangle} = \begin{cases} 0 & \text{if } u \neq v \\ 2^n & \text{if } u = v \end{cases}.$$

Proof. This can be seen as a result of the fact that the Walsh transform preserves orthogonality, as the sum is the inner product of the Walsh transforms of two complex-valued functions which are zero in all $\mathbb{F}_2^n$ and 1 at u or v. It can also be proven directly. If we consider $u = v$, then $\langle u, x\rangle + \langle v, x\rangle = 0$ for all $x \in \mathbb{F}_2^n$, from which we deduce the sum is 2^n. If $u \neq v$, then $\langle u, x\rangle + \langle v, x\rangle = 0$ exactly for the x which satisfy the linear equation $\langle u \oplus v, x\rangle = 0$. This equation is satisfied by the vectors of a hyperplane of $\mathbb{F}_2^n$, so there are exactly 2^{n-1} solutions. The sum is thus 0 since half of $u \in \mathbb{F}_2^n$ contribute to the sum with -1 and half contribute with 1. □

The following result gives us the Walsh transform of all linear and affine maps:

Proposition 3.13 (Equation 23 in [DGV94]). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be an affine vectorial Boolean function of the form $f(x) = Lx \oplus c$, where $L \in \mathrm{GL}(\mathbb{F}_2^n, \mathbb{F}_2^m)$ is a linear map and $c \in \mathbb{F}_2^m$ is a constant vector. Then*

$$\widehat{f}(u,v) = \begin{cases} 0 & \text{if } u \neq L^T v \\ (-1)^{\langle v,c\rangle} 2^n & \text{if } u = L^T v \end{cases} \quad \text{for all } u \in \mathbb{F}_2^n,\ v \in \mathbb{F}_2^m. \tag{3.31}$$

Proof. This can be deduced from the previous result and the fact that $\langle x, Ly\rangle = \langle L^T x, y\rangle$:

$$\begin{aligned} \widehat{f}(u,v) &= \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x\rangle \oplus \langle v, Lx \oplus c\rangle} \\ &= (-1)^{\langle v,c\rangle} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x\rangle \oplus \langle L^T v, x\rangle} = \begin{cases} 0 & \text{if } u \neq L^T v \\ (-1)^{\langle v,c\rangle} 2^n & \text{if } u = L^T v \end{cases}. \end{aligned}$$

□

In particular, this means the following:

- The Walsh transform of the addition of a constant c is a diagonal matrix, where the i-th element of the diagonal is equal to $(-1)^{\langle i,c\rangle}$. This, for example, describes the correlation matrix of a key addition step in a block cipher.
- The Walsh transform of a linear map is a matrix of zeroes and ones, and there is exactly one 1 in each column. The correlation matrix of a linear layer in a block cipher is thus a permutation matrix.

Also of interest to the cryptanalysis of block ciphers is the following result, which gives us the Walsh transform of the Sbox layer of a block cipher:

Proposition 3.14 (Equation 24 in [DGV94]). *Let $f_i : \mathbb{F}_2^{n_i} \longrightarrow \mathbb{F}_2^{m_i}$, $i = 1, \ldots, d$ be a collection of d vectorial Boolean functions. We consider the* bricklayer function *$\mathbf{F} : \mathbb{F}_2^{\sum_i n_i} \longrightarrow \mathbb{F}_2^{\sum_i m_i}$, which is obtained by concatenating these maps, that is, we take $\mathbf{F}(x_1|\cdots|x_d) = (f_1(x_1)|\cdots|f_d(x_d))$. Then, for any $(u_1, \ldots, u_d)$ in $\mathbb{F}_2^{n_1} \times \cdots \times \mathbb{F}_2^{n_d}$ and $(v_1, \ldots, v_d)$ in $\mathbb{F}_2^{m_1} \times \cdots \times \mathbb{F}_2^{m_d}$, we have*

$$\widehat{\mathbf{F}}((u_1|\cdots|u_d),(v_1|\cdots|v_d)) = \prod_{i=1}^{d} \widehat{f}_i(u_i, v_i). \tag{3.32}$$

Note that if $m_i = 1$, then $\widehat{f}_i(u_i, v_i) = \begin{cases} \widehat{f}_i(u_i) & \text{if } v_i = 1 \\ 0 & \text{if } u_i \neq 0, v_i = 0 \\ 2^{n_i} & \text{if } u_i = 0, v_i = 0 \end{cases}$.

Proof. This result can be deduced by a simple calculation:

$$\begin{aligned} &\widehat{\mathbf{F}}((u_1|\cdots|u_d),(v_1|\cdots|v_d)) = \\ &= \sum_{(x_1,\ldots,x_d) \in \mathbb{F}_2^{n_1} \times \cdots \times \mathbb{F}_2^{n_d}} (-1)^{\langle (u_1,\ldots,u_d),(x_1,\ldots,x_d)\rangle + \langle (v_1,\ldots,v_d),(f_1(x_1),\ldots,f_d(x_d))\rangle} \\ &= \sum_{x_1 \in \mathbb{F}_2^{n_1}} \cdots \sum_{x_d \in \mathbb{F}_2^{n_d}} (-1)^{\langle u_1,x_1\rangle + \langle v_1, f_1(x_1)\rangle} \cdot \ldots \cdot (-1)^{\langle u_d,x_d\rangle + \langle v_d, f_d(x_d)\rangle} \\ &= \prod_{i=1}^{d} \left(\sum_{x_i \in \mathbb{F}_2^{n_i}} (-1)^{\langle u_i,x_i\rangle + \langle v_i, f_i(x_i)\rangle} \right) = \prod_{i=1}^{d} \widehat{f}_i(u_i, v_i). \end{aligned}$$

□

This means that the correlation matrix C^F of the Sbox layer of a block cipher is the Kronecker product $C^{(S_1)} \otimes \cdots \otimes C^{(S_d)}$ of the correlation matrices of the Sboxes.

Finally, we provide the following result, which is equivalent to Proposition 2.20, and states that the Walsh transform of a composition of vectorial Boolean functions is the matrix product of their Walsh transforms:

Proposition 3.15 (Equation 15 in [DGV94]). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^l$ and $g : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^m$ be vectorial Boolean functions. The Walsh transform of the composition $g \circ f$ is the matrix product of the transforms of f and g:*

$$2^l \widehat{g \circ f}(u,v) = \sum_{w \in \mathbb{F}_2^l} \widehat{f}(u,w) \cdot \widehat{g}(w,v) \tag{3.33}$$

Proof. This result is another consequence of Lemma 3.12:

$$\begin{aligned}
\sum_{w\in\mathbb{F}_2^l} \widehat{f}(u,w)\cdot\widehat{g}(w,v) &= \sum_{w\in\mathbb{F}_2^l}\left(\sum_{x\in\mathbb{F}_2^n}(-1)^{\langle u,x\rangle+\langle w,f(x)\rangle}\right)\cdot\left(\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle w,z\rangle+\langle v,g(z)\rangle}\right)\\
&= \sum_{w\in\mathbb{F}_2^l}\sum_{x\in\mathbb{F}_2^n}\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle u,x\rangle+\langle w,f(x)\rangle+\langle w,z\rangle+\langle v,g(z)\rangle}\\
&= \sum_{x\in\mathbb{F}_2^n}\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle u,x\rangle+\langle v,g(z)\rangle}\underbrace{\sum_{w\in\mathbb{F}_2^l}(-1)^{\langle w,f(x)\rangle+\langle w,z\rangle}}_{2^l \text{ if } z=f(x),\ 0 \text{ otherwise}}\\
&= 2^l\sum_{x\in\mathbb{F}_2^n}(-1)^{\langle u,x\rangle+\langle v,g(f(x))\rangle} = 2^l\widehat{g\circ f}(u,v).
\end{aligned}$$

□

Propositions 3.13-3.15 can be used to describe the Walsh transforms of symmetric primitives based on Sboxes, linear transformations, and key xoring.

Part II

CONTRIBUTIONS

Chapter 4

Pruning the Fast Walsh Transform

The purpose of this chapter is to showcase my Walsh transform pruning technique for affine subspaces. An early, limited version was included in [FN20], and a generalised version which works on arbitrary affine subspaces was described in [Fló22]. More specifically, we develop algorithms which can evaluate the Walsh transform efficiently when it is known that a large fraction of input coordinates are zero, when only a small amount of outputs needs to be evaluated, or both. This pruning problem for the Walsh transform appears naturally when applying it to linear key recovery attacks: the specificities of the cipher construction, the data, and the key schedule may restrict the inputs and the outputs of the necessary Walsh transforms in ways the attacker may desire to exploit, but doing so requires careful analysis of the fast Walsh transform algorithm. Despite the motivation of linear cryptanalysis, I have decided to cover this topic in a separate chapter, as the pruned algorithm is of interest on its own and may find applications outside the scope of symmetric cryptology.

Contents

The chapter opens with a brief formal description of the pruning problem and how it's been tackled by previous publications in the case of the "classical" FFT. Section 4.2 covers the main subject of this chapter, which is a solution to the pruning problem when both the inputs and the outputs are restricted to affine subspaces of $\mathbb{F}_2^n$. We provide three versions of the algorithm, each one of which generalises and extends the previous one. For example, the first one can only be applied if we wish to fix the values of certain bits, while the last one is effective on arbitrary affine subspaces. Finally, Section 4.3 covers some strategies which can be used in cases in which the inputs and outputs do not lie in affine subspaces of small dimension, and Section 4.4 discusses some potential directions for further research.

4.1 Problem Statement and Related Work

Broadly speaking, our objective is to speed up the Fast Walsh Transform algorithm when the non-zero inputs or desired outputs are limited to previously-known fixed subsets of $\mathbb{F}_2^n$ by removing unnecessary computations. An algorithm which computes the desired outputs with fewer computations than the "full" fast transform requiring $n2^n$ additions will be called a *pruned* fast Walsh transform algorithm. More formally, we are interested in algorithms which solve the following problem:

Definition 4.1 (Problem statement). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be any vector in $\mathbb{C}\mathbb{F}_2^n$. We assume that lists $L, M \subseteq \mathbb{F}_2^n$ are given, and that $f(x) = 0$ for all $x \in \mathbb{F}_2^n \setminus L$. The aim is to compute $\widehat{f}(u)$ for all $u \in M$ with as few operations as possible.*

If L and M are arbitrary subsets of $\mathbb{F}_2^n$ we will say the pruning problem is *unstructured*. If some information about L and M is known (for example, that they lie within some given affine subspaces of $\mathbb{F}_2^n$ of fixed dimension) we will say the pruning problem is *structured*.

4.1.1 Overview of Previous Results for the One-dimensional FFT

Before looking at the case of the Walsh transform, which is the multidimensional $(2,\dots,2)$-discrete Fourier transform, we will look at previous work in the case of the one-dimensional 2^n-discrete Fourier transform, that is, the "classical" discrete Fourier transform. This problem has been given more attention historically due to the popularity and prevalence of the one-dimensional FFT and due to the presence of many interesting applications in diverse fields.

The first pruned version of the FFT algorithm was given by Markel [Mar71], who pruned decimation-in-frequency algorithm for the case in which L consists of the first 2^r, $r < s$ points of the input, and $M = \mathbb{Z}/2^n\mathbb{Z}$. This is achieved by removing all operations from the algorithm which are performed exclusively on zeroes. The resulting algorithm has a time complexity $\mathcal{O}(r2^n)$, which means the number of operations is reduced by a factor of roughly r/n. An analogous pruned version of the decimation-in-time algorithm for the same lists L and M was described by Skinner [Ski76]. This algorithm achieves savings which are slightly better by removing some twiddle factor multiplications, but the improvement is still roughly r/n.

The first pruned algorithm to consider both zero inputs and a restricted set of outputs was given by Sreenivas and Rao [SR79]. It applies the same pruning approach to both the decimation-in-time and the decimation-in-frequency algorithms, and picks the one giving the smallest time complexity. It achieves a similar time complexity saving to the two previous algorithms. A pruned version decimation-in-time algorithm was given by Nagai [Nag86], focusing on the case in which $L = \mathbb{Z}/2^n\mathbb{Z}$ and M is any (possibly shifted) consecutive window of 2^r frequencies, and achieving the same complexity reduction of r/n.

An alternative approach to simple removal of operations called *transform decomposition* was introduced by Sorensen and Burrus [SB93]. The general idea of this technique is to map the given non-zero inputs to the transform of size 2^n to the inputs of a series of separate DFTs of smaller size. After these transforms are computed, the resulting outputs are combined to obtain the desired outputs. Although the approach works very well in the cases covered by previous works and can also be applied in other cases, it is not very effective in the latter.

All the aforementioned algorithms exhibit similar complexities: in essence, evaluating 2^r points of a 2^n point transform costs $\mathcal{O}(r2^n)$, which means the complexity was reduced by a factor of around r/n. However, an interesting case was found by Shousheng and Torkelson [ST96]: when the list M is a *comb* of equidistant points as opposed to the continuous bands considered in previous literature, a smaller complexity of $\mathcal{O}(2^n + r2^r)$ can be achieved.

Although the basic unnecessary computation removal used to construct the algorithms of these works can be applied to unstructured, arbitrary input and output sets, the idea was formalised as a general algorithm by Alves et al. [AOS00]. Given input and output lists L and M, the pruned FFT algorithm is constructed using a *traceback* technique. A more careful complexity analysis was also carried out, and in particular a formula for the average complexity as a function of $n, |L|$ and $|M|$ was given. An equivalent technique was (re)introduced by Hu and Wan [HW05]. The efficiency of the overhead computations required for the traceback was improved in [SS05]. Similar generalised pruning techniques have also been proposed for the case of mixed-radix and composite-length DFTs in works such as [WZSL12, CMTY15].

In the remainder of this chapter, we tackle the pruning problem in the case of the Walsh transform or $(2,\dots,2)$-dimensional DFT. We mostly focus on the case in which L and M lie in affine subspaces of $\mathbb{F}_2^n$. The pruning problem for the Walsh transform has some fundamental differences to the equivalent problem for the one-dimensional FFT. We note that in the case of the Walsh transform, it is possible to make use of the vector space structure of $\mathbb{F}_2^n$, and in particular we can invoke the properties of the inner product $\langle\cdot,\cdot\rangle$, while for the standard FFT only the structure of the cyclic group $\mathbb{Z}/2^n\mathbb{Z}$ can be used. As a result, the pruning techniques introduced in this chapter are not compatible with the one-dimensional FFT (or at least not trivially so), and different from the ones featured in previous works on the topic.

The Walsh transform pruning problem and has seen some attention in the literature, in particular Jankovic et al. [JSD01] introduced a binary decision diagram-based method for computing the Walsh transform to provide solutions to some Walsh transform output pruning problems. However, this approach has the disadvantage that the number of required operations isn't just dependent on the structure of the output bins, but also on the input function. A related problem to the pruning one is the sparse transform problem, where we assume that the output has a certain sparsity (in other words, the number of non-zero positions in the output is bound by some constant), which was tackled in [CI17].

4.2 Affine Pruning Algorithms for the Fast Walsh Transform

This section focuses on the following structured pruned Walsh transform problem:

Definition 4.2 (Affine pruning problem). *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a complex vector of length 2^n. We assume we are given lists $L, M \subseteq \mathbb{F}_2^n$ as well as vector subspaces $X, U \subseteq \mathbb{F}_2^n$ and vectors $x_0, u_0 \in \mathbb{F}_2^n$ so that $L \subseteq x_0 + X, M \subseteq u_0 + U$, and $f(x) = 0$ for all $x \notin L$. The aim is to compute $\widehat{f}(u)$ for all $u \in M$ with as few operations as possible.*

The only added assumption with respect to the unstructured problem is that we consider that the subsets L and M are entirely contained inside some affine subspaces $x_0 + X$ and $u_0 + U$ whose dimension is significantly smaller than the dimension of the ambient space, n.

In order to better understand how the affine Walsh transform pruning problem was solved, we will describe three different attempts. The first was published in [FN20] and can only be applied when X and U are generated by some standard unit vectors e_i of Hamming weight 1. The second version is an unpublished attempt to generalise the previous approach by associating a fast Walsh transform algorithm to each basis of $\mathbb{F}_2^n$ which fulfils some orthogonality properties. These restrictions on the basis, however, mean that the resulting pruned algorithm is not applicable to arbitrary affine subspaces. We finally describe the algorithm shown in [Fló22], which relies on reducing the transform to one of smaller dimension, and which we show is effective and efficient for all choices of affine subspaces of the input and output indices.

4.2.1 Approach 1: Fixing Values of Bits and Commutativity

We will start tackling the affine pruning problem on the Walsh transform by discussing the partial solution proposed in [FN20]. This technique can be applied when either the inputs or the outputs to the transform have some bits of their binary indices restricted to some fixed values. In terms of the formal affine pruning problem, we consider the case in which either X or U is generated by some standard basis vectors e_i.

We will initially focus on the case $X = \mathbb{F}_2^n$ (so only the outputs are restricted). In this case, it may seem more intuitive to think of the problem in terms $U^\perp$ instead of U, as $U^\perp$ consists of the span of the bits whose values are fixed, and U consists of the span of the bits which are free. The specific values of the fixed bits are then provided by a vector $u_0 \in U^\perp$ (this condition ensures that all the free bits of u_0 are set to 0).

The general idea of this pruning strategy is to apply the traceback technique of [AOS00] on the fast Walsh transform algorithm generated using a well-chosen permutation $\pi \in S_n$. The importance of the choice of π is highlighted by the examples given in Figure 4.1, where we consider that $U^\perp = \operatorname{span}\{e_1, e_3\}$ and $u_0 = (0, 0, 1, 0)$.

The cost of the full fast Walsh transform algorithm of size 16 is $2 \cdot 2^4 = 64$ complex additions and subtractions. If we remove the unnecessary computations from the "standard" algorithm given by the permutation $\pi_1 = \mathrm{id}$, we obtain a pruned version which requires 36 additions. If we do the same thing for the algorithm generated by $\pi_2 = (4, 2, 3, 1)$, this amount decreases to just 20 operations, almost halving the complexity.

So, in order to find a good solution to the pruning problem, we must solve that of finding a good permutation π for a given pruning problem. Given $U^\perp$ and u_0, we wish to find an optimal permutation π and provide a compact formula for the resulting time complexity in terms of parameters such as the size of the input vector and the dimension of $U^\perp$.

Let us look at the example again to see how both permutations interact differently with the desired outputs. In particular, we note that in the case of π_1, when we trace back through the last set of butterflies, the number of active wires grows from 4 to 8. In the case of π_2, the number of active wires before the last stage is still 4, because the elements of $M = u_0 + U$ are paired with each other in that stage's butterflies. The same thing happens when we undo the third stage of the transform. This suggests that we should leave for the end of the transform the stages in which elements of $u_0 + U$ are paired with each other in the butterflies.

Proposition 4.3. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a complex vector of length 2^n. We are given a list $L \subseteq \mathbb{F}_2^n$ so that $f(x) = 0$ for all $x \notin L$. We are also given a vector subspace $U \subseteq \mathbb{F}_2^n$ of dimension s in the form $U = \operatorname{span}\{e_{i-1} : i \in \mathcal{S}\}$, $\mathcal{S} \subseteq \{1, \ldots, n\}$, $|\mathcal{S}| = \dim(U) = s$ and a vector $u_0 \in \mathbb{F}_2^n$. It is possible to compute $\widehat{f}(u)$ for all $u \in u_0 + U$ with exactly*

$$\begin{aligned} |L| + s2^s &\text{ additions and subtractions,} \\ 2^n + (s-1)2^s &\text{ in the case } L = \mathbb{F}_2^n. \end{aligned} \tag{4.1}$$

Note that, given $U^\perp = \operatorname{span}\{e_{i-1} : i \notin \mathcal{S}\}$ instead of U, we can substitute s for $n - \dim\left(U^\perp\right)$.

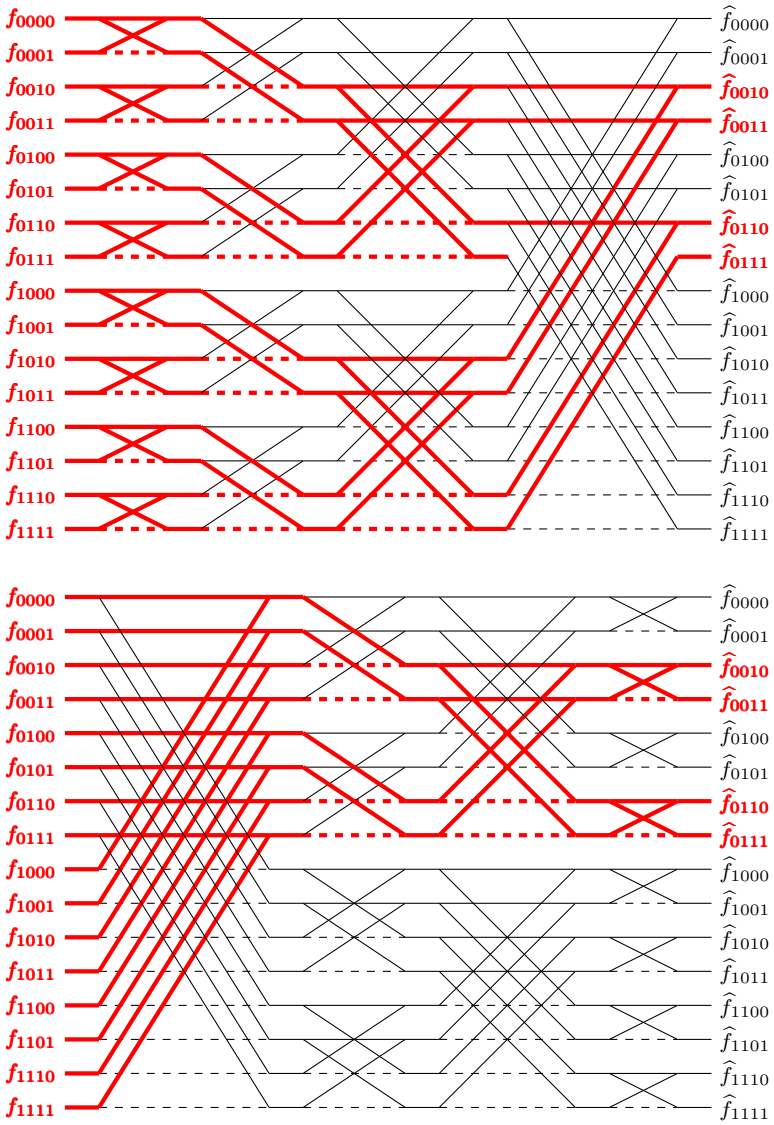


Figure 4.1: The result of applying the traceback pruning technique on the fast Walsh transform of size $2^4 = 16$ when the output coordinates are restricted according to the conditions $u_1 = 1,\ u_3 = 0$. We have considered the fast Walsh transform algorithms associated to the permutations $\pi_1 = \mathrm{id} = (1, 2, 3, 4)$ and $\pi_2 = (4, 2, 3, 1)$. We conclude that for π_1, the resulting pruned Walsh transform algorithm requires 36 complex additions, while in the case of π_2 we only need 20.

Proof. We will apply the traceback pruning approach on the fast Walsh transform algorithm generated by any permutation $\pi = (\pi(1), \ldots, \pi(n))$ which satisfies the (equivalent) conditions

$$\begin{aligned} \{\pi(1), \ldots, \pi(n-s)\} &= \{1, \ldots, n\} \setminus \mathcal{S}, \\ \{\pi(n-s+1), \ldots, \pi(n)\} &= \mathcal{S}. \end{aligned}$$

Let us compute the number of required operations when we prune a fast Walsh transform of this form. We will consider the first $n-s$ stages and the last s stages separately.

- We start the traceback from the end of the transform. At the output of the transform, we have 2^s positions whose value we are interested in. Since at the i-th stage, the positions which are paired for the butterflies differ by $e_{\pi(i)-1}$ and we have chosen π so that $e_{\pi(i)-1} \in U$, we know that all the butterflies pair elements of $u_0 + U$ with each other. This means that, at each stage, we are interested in the values of 2^s positions, and we will compute 2^s complex additions and subtractions. The total cost for the s stages will thus be $s2^s$ additions and subtractions.

- In the first $n-s$ stages, we deal with the opposite situation: every butterfly pairs a position which is active with one which is not. This means that the number of active wires doubles with each stage until the full input is active. The total cost of these stages in additions is $\sum_{i=1}^{n-s} 2^{n-i} = 2^n - 2^s$. Furthermore, if the list L is significantly smaller than $\mathbb{F}_2^n$, we can go over the list ignoring all the other inputs, which means we can perform this part with at most L additions and subtractions.

We obtain the full cost by adding the number of operations in the two stages. □

We note that this complexity formula has two major implications. First, when s is comparatively small with respect to n, the complexity becomes dominated by the first term 2^n. This means that even if all the inputs are active, we have effectively removed the logarithmic factor from the classical $n2^n$ complexity formula. Second, if we also have $|L| \ll 2^n$, the cost of the pruned transform becomes fully independent of its full size, and depends only on the number of active inputs and outputs.

The previous result can be adapted to the case in which the inputs are restricted instead of the outputs, and even the case in which bit restrictions are applied to both the input and the output. We get the following results, which are given with proof sketches:

Proposition 4.4. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a complex vector of length 2^n. We are given a vector subspace $X \subseteq \mathbb{F}_2^n$ of dimension r in the form $X = \operatorname{span}\{e_i : i \in \mathcal{R}\}$, $\mathcal{R} \subseteq \{0, \ldots, n-1\}$, $|\mathcal{R}| = \dim(X) = r$ and a vector $x_0 \in \mathbb{F}_2^n$ so that $f(x) = 0$ for all $x \notin x_0 + X$. We are also given a list $M \subseteq \mathbb{F}_2^n$. It is possible to compute $\widehat{f}(u)$ for all $u \in M$ with exactly*

$$\begin{aligned} &r2^r \text{ additions and } |M| \text{ memory accesses,} \\ &r2^r \text{ additions and } 2^n - 2^r \text{ memory accesses in the case } M = \mathbb{F}_2^n. \end{aligned} \tag{4.2}$$

Proof. We will show how an appropriate permutation π may be chosen, and the rest of the proof should follow similar arguments to the ones we gave for Proposition 4.3. We can use any permutation $\pi = (\pi(1), \ldots, \pi(n))$ which fulfils the conditions:

$$\begin{aligned} \{\pi(1), \ldots, \pi(r)\} &= \mathcal{R} \\ \{\pi(r+1), \ldots, \pi(n)\} &= \{1, \ldots, n\} \setminus \mathcal{R} \end{aligned}$$

The main difference with the reasoning used in Proposition 4.3 is that the first r stages consist of 2^r additions and subtractions each, and the rest consist of successive copying of memory registers. □

Proposition 4.5. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a complex vector of length 2^n. We are given a vector subspace $X \subseteq \mathbb{F}_2^n$ of dimension r in the form $X = \operatorname{span}\{e_{i-1} : i \in \mathcal{R}\}$, $\mathcal{R} \subseteq \{1, \ldots, n\}$, $|\mathcal{R}| = r$ and a vector $x_0 \in \mathbb{F}_2^n$ so that $f(x) = 0$ for all $x \notin x_0 + X$. We are also given a vector subspace $U \subseteq \mathbb{F}_2^n$ of dimension s in the form $U = \operatorname{span}\{e_{i-1} : i \in \mathcal{S}\}$, $\mathcal{S} \subseteq \{1, \ldots, n\}$, $|\mathcal{S}| = s$ and a vector $u_0 \in \mathbb{F}_2^n$. We also assume that $X \cap U$ has dimension $t = |\mathcal{R} \cap \mathcal{S}|$. It is possible to compute $\widehat{f}(u)$ for all $u \in u_0 + U$ with exactly*

$$2^r + (t-1)2^t \text{ additions and } 2^s - 2^t \text{ memory accesses.} \tag{4.3}$$

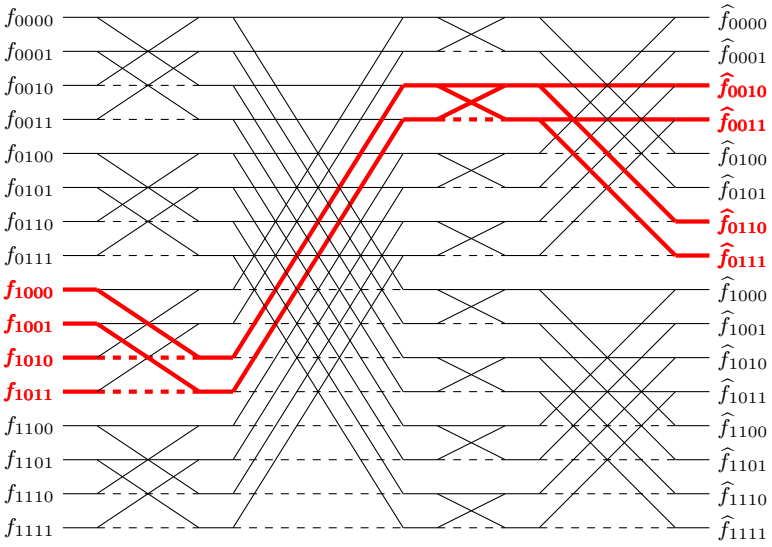


Figure 4.2: Example of simultaneous pruning of both inputs and outputs. In the case $X = \operatorname{span}\{e_0, e_1\}, x_0 = (1,0,0,0)$ and $U = \operatorname{span}\{e_0, e_2\}, u_0 = (0,0,1,0)$, we can compute all the desired outputs with just 4 additions and 2 memory accesses by taking $\pi = (2,4,1,3)$. In this case the cost of computing the outputs directly from the definition is 16 operations.

Proof. We will show how an appropriate permutation π may be chosen, and the rest of the proof should follow similar arguments to the ones we gave for Proposition 4.3. We can use any permutation $\pi = (\pi(1), \ldots, \pi(n))$ which fulfils the conditions:

$$\begin{aligned} \{\pi(1), \ldots, \pi(r-t)\} &= \mathcal{R} \setminus \mathcal{S}, \\ \{\pi(r-t+1), \ldots, \pi(n-s+t)\} &= (\{1, \ldots, n\} \setminus (\mathcal{R} \cup \mathcal{S})) \cup (\mathcal{R} \cap \mathcal{S}), \\ \{\pi(n-s+t+1), \ldots, \pi(n)\} &= \mathcal{S} \setminus \mathcal{R}. \end{aligned}$$

□

4.2.2 Approach 2: Choosing Advantageous Basis

We will now describe an attempt at generalising the pruning approach which has been exposed for the fixed bit values case to more general affine subspaces. Although it is largely superseded in both generality and simplicity by the different approach which is introduced in the next section, I believe it can still provide some insight on the nature of the fast Walsh transform algorithm and the key importance of its algebraic structure, and that some of these ideas might eventually prove useful in future pruning techniques.

The idea behind this generalisation is the following. In the previous approach, we leveraged the fact that we can pick any permutation $\pi \in S_n$ to obtain an associated fast Walsh transform algorithm. By choosing π carefully, we can obtain an algorithm which prunes well for a given restriction on the inputs and outputs. The permutation $\pi = (\pi(1), \ldots, \pi(n))$ determines an ordering of the standard basis $\{e_0, \ldots, e_{n-1}\}$, so that at the i-th stage of the transform, butterflies are applied on pairs of entries which differ by $e_{\pi(i)-1}$ in their coordinates. This observation is followed by the question: is it possible to do the same thing for other basis? This would broaden the choices of fast Walsh transform algorithm when X and U are arbitrary, increasing the likelihood that an algorithm with better complexity exists.

Definition 4.6. *Let $\mathcal{B} = \{b_0, \ldots, b_{n-1}\} \subseteq \mathbb{F}_2^n$ be a basis of $\mathbb{F}_2^n$. We will say $\mathcal{B}$ is an* orthonormal basis *if $\langle b_i, b_j \rangle = \delta_{ij}$. From here and until the remainder of this section, we will consider basis are* ordered*, that is, a basis $\mathcal{B} = (b_0, \ldots, b_{n-1})$ will be an n-tuple of vectors of $\mathbb{F}_2^n$ whose elements form a basis of $\mathbb{F}_2^n$ in the traditional, non-ordered sense.*

Please note that the notion of orthogonality in $\mathbb{F}_2^n$ behaves quite differently from orthogonality in real or complex vector spaces, as the inner product in $\mathbb{F}_2^n$ is not definite, and there are vectors which are orthogonal to themselves. In particular, this means that there is no equivalent to the Gram-Schmidt orthonormalisation algorithm in binary vector spaces.

Proposition 4.7. *Let $\mathcal{B} = (b_0, \ldots, b_{n-1})$ be an ordered orthonormal basis of $\mathbb{F}_2^n$, and let $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ be a complex function on $\mathbb{F}_2^n$. The Walsh transform of f can be evaluated using the following formula:*

$$\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u, b_0 \rangle \langle b_0, x \rangle + \cdots + \langle u, b_{n-1} \rangle \langle b_{n-1}, x \rangle}. \tag{4.4}$$

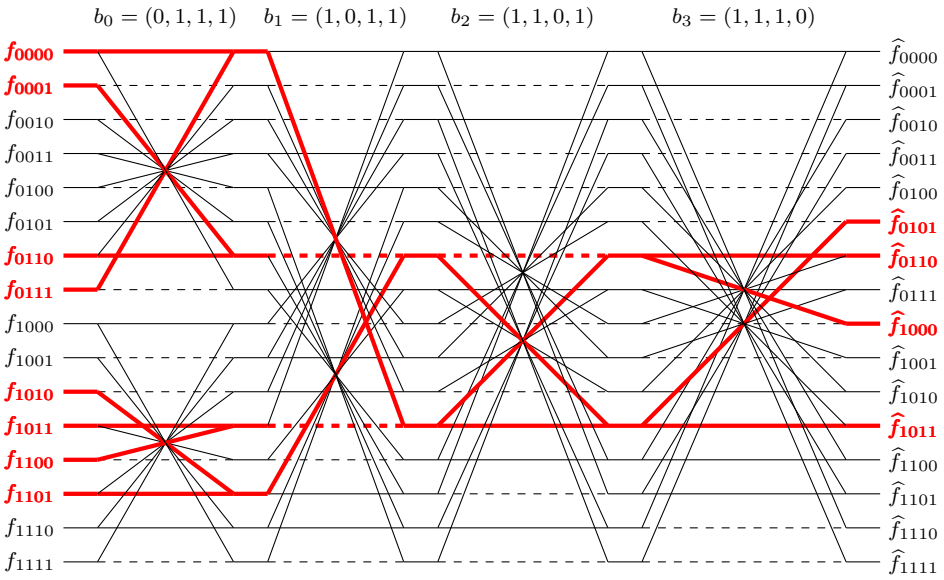


Figure 4.3: An example of generalised fast Walsh transform algorithm associated to the basis $\mathcal{B} = (b_0, b_1, b_2, b_3) = ((0,1,1,1),(1,0,1,1),(1,1,0,1),(1,1,1,0))$. We also show how it provides an effective pruned transform when $X = \operatorname{span}\{(0,0,0,1),(0,1,1,0),(1,1,0,0)\}$, $x_0 = (0,0,0,0)$ and $U = \operatorname{span}\{(1,1,1,0),(0,0,1,1)\}$, $u_0 = (0,1,0,1)$. This is possible because $X \cap U = \operatorname{span}\{b_2\}$, $X = \operatorname{span}\{b_0, b_1, b_2\}$, and $U = \operatorname{span}\{b_2, b_3\}$. The pruned transform requires only 8 additions and 2 memory accesses.

This formula can be split in a manner analogous to Equation 3.18:

$$\begin{aligned}\widehat{f}(u) = &\sum_{\substack{x\in\mathbb{F}_2^n \\ \langle b_{n-1},x\rangle=0}} (-1)^{\langle u,b_0\rangle\langle b_0,x\rangle+\cdots+\langle u,b_{n-2}\rangle\langle b_{n-2},x\rangle} f(x) \\ + (-1)^{\langle u,b_{n-1}\rangle} &\sum_{\substack{x\in\mathbb{F}_2^n \\ \langle b_{n-1},x\rangle=1}} (-1)^{\langle u,b_0\rangle\langle b_0,x\rangle+\cdots+\langle u,b_{n-2}\rangle\langle b_{n-2},x\rangle} f(x),\end{aligned} \tag{4.5}$$

and iterating this operation leads to a fast algorithm for the Walsh transform.

Proof. This formula can be seen as a consequence of the definition of the Walsh transform as the discrete Fourier transform on the group $\mathbb{F}_2^n$, which doesn't privilege the standard basis of $\mathbb{F}_2^n$ over other orthonormal basis. However, we can also easily deduce it from the "classical" Walsh transform formula. If we write $u = \sum_{j=0}^{n-1} u_j^{\mathcal{B}} b_j$ as a linear combination of the vectors of $\mathcal{B}$, we have $\langle u, b_i\rangle = \sum_{j=0}^{n-1} u_j^{\mathcal{B}} \langle b_j, b_i\rangle = u_i^{\mathcal{B}}$. From this, we conclude that, given $x \in \mathbb{F}_2^n$, we have

$$\langle u, x\rangle = \left\langle \sum_{i=0}^{n-1} b_i^{\mathcal{B}} u_i, x \right\rangle = \sum_{i=0}^{n-1} b_i^{\mathcal{B}} \langle b_i, x\rangle = \sum_{i=0}^{n-1} \langle u, b_i\rangle\langle b_i, x\rangle,$$

from which the desired expression is quickly obtained. The split version of the formula is obtained by separating the terms in the sum for which $\langle b_{n-1}, x\rangle = 0$ and the terms for which $\langle b_{n-1}, x\rangle = 1$ and substituting the value of $\langle b_{n-1}, x\rangle$ in the exponent. □

From this result, we can construct the generalised fast Walsh transform algorithm (Algorithm 9). As a parameter, the algorithm takes any orthonormal basis of $\mathbb{F}_2^n$, and each stage of the transform applies butterflies on the pairs of entries which differ by one of the elements of the basis. It is very important that $\mathcal{B}$ is orthonormal, and if this is not the case, the algorithm doesn't output the Walsh transform of f. This algorithm is a generalisation because the previous version which takes a permutation $\pi \in S_n$ corresponds to giving this new algorithm the orthonormal ordered basis $\mathcal{B} = \left(e_{\pi(1)-1}, \ldots, e_{\pi(n)-1}\right)$.

We now investigate how this broader collection of fast Walsh transform algorithms could potentially be used to obtain better pruning techniques. Figure 4.3 illustrates an example in which very good results can be obtained. In this specific case, we see that there exists an orthonormal basis $\mathcal{B}$ so that X, U and $X \cap U$ are all generated by some elements of $\mathcal{B}$. This means we fall into a situation very similar to that of Proposition 4.5, and we can obtain the same complexity given the dimensions of X, U and $X \cap U$. Let

Algorithm 9: Fast Walsh transform using an arbitrary orthonormal basis

Parameters: $\mathcal{B} = (b_0, \ldots, b_{n-1})$ an ordered orthonormal basis of $\mathbb{F}_2^n$.
Input: $f : \mathbb{C}\mathbb{F}_2^n \longrightarrow \mathbb{C}$.
Output: $\widehat{f} : \mathbb{C}\mathbb{F}_2^n \longrightarrow \mathbb{C}$.
for $k \leftarrow 1$ **to** n **do** // The FWT "stages"
  foreach $i \in \mathbb{F}_2^n$ *with* $\langle i, b_{k-1} \rangle = 0$ **do** // The "butterflies"
    $j \leftarrow i + b_{k-1}$;
    $(f(i), f(j)) \leftarrow (f(i) + f(j), f(i) - f(j))$;
  end
end
return f;

Y and V be subspaces so that X is the direct sum of $X \cap U$ and Y and U is the direct sum of $X \cap U$ and V. An efficient pruned fast Walsh transform can be obtained from an algorithm generated by reordering the basis $\mathcal{B} = (b_0, \ldots, b_{n-1})$ so that

$$\operatorname{span}\{b_0, \ldots, b_{r-t-1}\} = Y,$$
$$\operatorname{span}\{b_{n-s-1}, \ldots, b_{n-1}\} = V.$$

In practice and in order to make the traceback step a bit easier, we can perform a change of basis both at the beginning and the end of the transform on the coordinates of the bins which transforms $\mathcal{B}$ into the standard basis, so that the inner stages of the transform become the standard Walsh transform with $\pi = \mathrm{id}$.

This approach, however, becomes rather difficult to use for arbitrary affine subspaces. The main obstacle is the existence of self-orthogonal subspaces, in which all vectors are orthogonal to each other, including themselves. This means such a subspace can never be written as the span of some vectors of an orthonormal basis. Even if we ignore such subspaces, this method still involves some complicated steps. We also have no resulting complexity formula which takes all the parameters into account (the dimensions r, s and t are not sufficient as they do not provide orthogonality information).

As we will see in the next section, we can find a method which is proven to work for arbitrary subspaces (including self-orthogonal ones), with arguably simpler linear algebra, and performing most of the calculations inside a standard fast Walsh transform routine. Furthermore, we have provided a closed formula for the complexity of this algorithm and shown that it's close to optimal. For these reasons, we believe that the algorithm we describe in the next section is more desirable than one obtained through generalised fast Walsh transforms, at least for the applications that we consider in the scope of this thesis.

4.2.3 Approach 3: General Affine Pruning

We will now describe a different pruning approach for the Walsh transform than the two considered in the previous pages, and which is finally able to solve Problem 4.2 for all possible parameters, as introduced in [Fló22]. In fact, we can prove that this algorithm is essentially optimal if we assume the optimality of the standard Walsh transform.

The motivation behind this approach is the behaviour we have observed with the two previous attempts: in all the cases we have considered so far, the algorithm consists of three basic steps:

1. All the non-zero inputs are condensed into a shorter vector of length 2^t. Each input is added to a single position of the shorter vector, and may have its sign swapped.

2. A standard Walsh transform of length 2^t is applied on the shortened vector.

3. The outputs of the small transform are expanded into the desired outputs of the full transform with just register copying and sign swaps.

Since we keep running into pruned algorithms which operate like this, we decided to try and build such an algorithm directly, instead of considering a full fast transform from which unnecessary operations have been removed. Let us first consider a concrete example.

Example. We again consider the Walsh transform of size $16 = 2^4$. The full fast Walsh transform algorithm requires $4 \cdot 2^4 = 64$ additions and subtractions in total. We consider a pruning problem in the setting

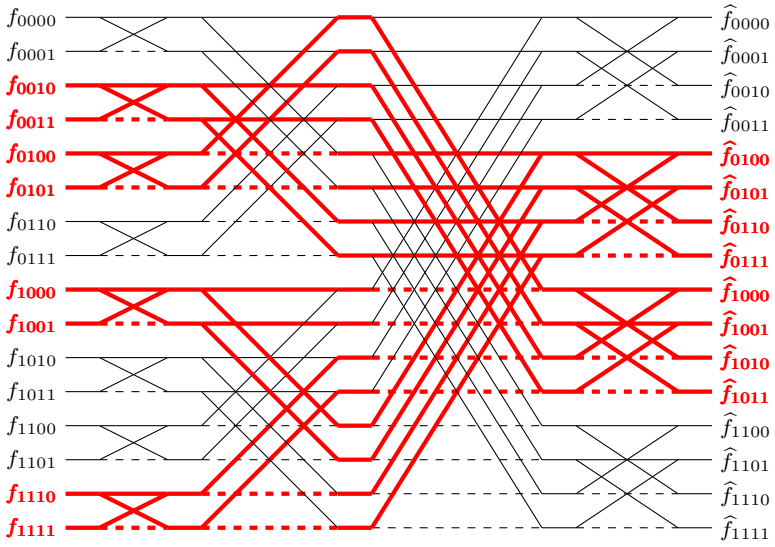


Figure 4.4: A pruning attempt using traceback on a carefully-chosen ordering of the standard basis. It requires a total of 24 additions/subtractions and 8 register copies, in contrast to the 64 operations required for the computation of the full transform.

of 4.2 with the input and output lists $L = x_0 + X$ and $M = u_0 + U$ equal to

$$\begin{aligned} x_0 &= (0,0,1,0), \quad X = \operatorname{span}\{(0,0,0,1),\ (0,1,1,0),\ (1,0,1,0)\}, \\ u_0 &= (0,1,0,0), \quad U = \operatorname{span}\{(0,0,0,1),\ (0,0,1,0),\ (1,1,0,0)\}. \end{aligned}$$

We first attempt to apply the traceback-based pruning approach on an appropriately chosen standard basis ordering. Since $e_0 \in X$ and $e_0, e_1 \in U$, we have chosen the permutation $\pi = (1,3,4,2)$, and the results are shown in Figure 4.4. By removing unnecessary computations from the resulting fast Walsh transform, we obtain the desired outputs with a total of 32 operations.

If we try to apply the approach of the previous section, we find that no orthonormal basis exists fulfilling the ideal conditions which were outlined. We would like the basis $\mathcal{B} = (b_0, b_1, b_2, b_3)$ to satisfy $X = \operatorname{span}\{b_1, b_2, b_3\}$ and $U = \operatorname{span}\{b_0, b_1, b_2\}$. This in particular would mean that $X \cap U = \operatorname{span}\{b_1, b_2\}$. However, since we know that $X \cap U = \operatorname{span}\{(0,0,0,1),(1,1,0,0)\}$, we can check that no $b_1, b_2 \in X \cap U$ exist which are orthogonal to each other and not self-orthogonal.

Given this situation, we decide to try again from the beginning, and we look at the individual expressions for each of the desired outputs, which are reproduced in Figure 4.5. By examining these formulas, we observe the following properties:

$$\widehat{f}_{0100} = -\widehat{f}_{1010},\ \widehat{f}_{0101} = -\widehat{f}_{1011},\ \widehat{f}_{0110} = -\widehat{f}_{1000},\ \widehat{f}_{0111} = -\widehat{f}_{1001}$$

The difference in the indices in each of these pairs is $(1,1,1,0)$, which is orthogonal to X. We also note that there are pairs of inputs which always appear with opposite signs, namely (f_{0010}, f_{1110}), (f_{0011}, f_{1111}), (f_{0100}, f_{1000}), and (f_{0101}, f_{1001}). In this case, the difference between the indices is $(1,1,0,0)$, which is orthogonal to U.

This suggests an algorithm which subtracts the elements of these input pairs from one another at the beginning and duplicates the output pairs at the end, such as the one in Figure 4.6, which has a total cost of 12 additions and subtractions and 4 memory accesses. By arranging the intermediate values in the appropriate way, we can even write this algorithm in such a way that the central step is a fast Walsh transform itself.

We now proceed to formalise the "trick" we used in the example. We start by proving the following auxiliary lemma. In essence, it constructs basis of U and X which are orthogonal to each other, and which will be used to organise the inputs and the outputs.

Lemma 4.8. *Let $X, U \subseteq \mathbb{F}_2^n$ be vector subspaces of $\mathbb{F}_2^n$. We can define t as*

$$t := \dim\left(\frac{X}{X \cap U^\perp}\right) = \dim\left(\frac{U}{U \cap X^\perp}\right). \tag{4.6}$$

$$
\begin{array}{lcllllllll}
\widehat{f}_{0100} & = & +f_{0010} & +f_{0011} & -f_{0100} & -f_{0101} & +f_{1000} & +f_{1001} & -f_{1110} & -f_{1111} \\
\widehat{f}_{0101} & = & +f_{0010} & -f_{0011} & -f_{0100} & +f_{0101} & +f_{1000} & -f_{1001} & -f_{1110} & +f_{1111} \\
\widehat{f}_{0110} & = & -f_{0010} & -f_{0011} & -f_{0100} & -f_{0101} & +f_{1000} & +f_{1001} & +f_{1110} & +f_{1111} \\
\widehat{f}_{0111} & = & -f_{0010} & +f_{0011} & -f_{0100} & +f_{0101} & +f_{1000} & -f_{1001} & +f_{1110} & -f_{1111} \\
\widehat{f}_{1000} & = & +f_{0010} & +f_{0011} & +f_{0100} & +f_{0101} & -f_{1000} & -f_{1001} & -f_{1110} & -f_{1111} \\
\widehat{f}_{1001} & = & +f_{0010} & -f_{0011} & +f_{0100} & -f_{0101} & -f_{1000} & +f_{1001} & -f_{1110} & +f_{1111} \\
\widehat{f}_{1010} & = & -f_{0010} & -f_{0011} & +f_{0100} & +f_{0101} & -f_{1000} & -f_{1001} & +f_{1110} & +f_{1111} \\
\widehat{f}_{1011} & = & -f_{0010} & +f_{0011} & +f_{0100} & -f_{0101} & -f_{1000} & +f_{1001} & +f_{1110} & -f_{1111}
\end{array}
$$

Figure 4.5: The desired entries of $\widehat{f}$ written in terms of the non-zero entries of f.

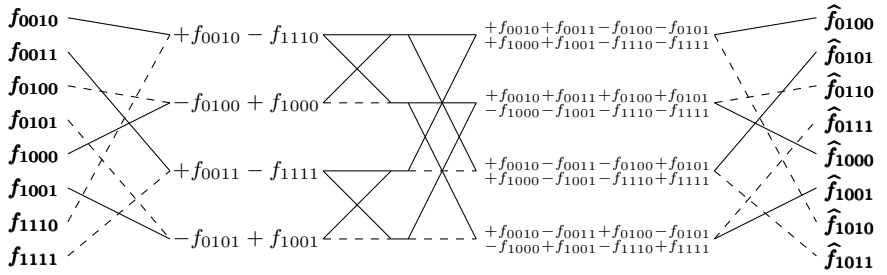


Figure 4.6: Organising the inputs and outputs carefully allows us to reduce the cost of the transform to just 16 operations.

Furthermore, the inner product $\langle y, v\rangle$ of an element y of $X/(X \cap U^\perp)$ and an element v of $U/(U \cap X^\perp)$ can be defined as the inner product of any two representatives. There also exist isomorphisms $\phi : X/(X \cap U^\perp) \xrightarrow{\cong} \mathbb{F}_2^t$ and $\psi : U/(U \cap X^\perp) \xrightarrow{\cong} \mathbb{F}_2^t$ which preserve it:

$$\langle y, v\rangle = \langle \phi(y), \psi(v)\rangle \text{ for all } y \in \frac{X}{X \cap U^\perp}, v \in \frac{U}{U \cap X^\perp}. \tag{4.7}$$

Equivalently, there exist basis $\{y_1, \ldots, y_t\}$ of $X/(X \cap U^\perp)$ and $\{v_1, \ldots, v_t\}$ of $U/(U \cap X^\perp)$ for which $\langle y_i, v_j\rangle = \delta_{ij}$ for all $i, j = 1, \ldots, t$.

Proof. The equality of the dimensions is a consequence of the dimension formula, the fact that $\dim(V)^\perp = n - \dim(V)$, and the properties of orthogonal spaces.

$$
\begin{aligned}
\dim\left(X/(X \cap U^\perp)\right) &= +\dim(X) - \dim\left(X \cap U^\perp\right) \\
&= +\, n - \dim\left(X^\perp\right) - n + \dim\left((X \cap U^\perp)^\perp\right) \\
&= -\dim\left(X^\perp\right) + \dim\left(X^\perp + U\right) \\
&= -\dim\left(X^\perp\right) + \dim\left(X^\perp\right) + \dim(U) - \dim\left(U \cap X^\perp\right) \\
&= +\dim\left(U/(U \cap X^\perp)\right).
\end{aligned}
$$

It is also easy to show that the inner product $\langle y, v\rangle$ is well-defined for any $y \in X/(X \cap U^\perp)$ and $v \in U/(U \cap X^\perp)$. Let $x_1, x_2 \in X$ and $u_1, u_2 \in U$ which verify $\overline{x_1} = \overline{x_2}$ and $\overline{u_1} = \overline{u_2}$ (that is, $x_1 + x_2 \in U^\perp$ and $u_1 + u_2 \in X^\perp$). Then

$$\langle x_1, u_1\rangle = \langle x_2, u_2\rangle + \langle x_1 + x_2, u_1\rangle + \langle x_2, u_1 + u_2\rangle = \langle x_2, u_2\rangle.$$

We will construct a pair of "orthogonal" bases starting from two arbitrary bases $\{y_1, \ldots, y_t\}$ of $X/(X \cap U^\perp)$ and $\{v_1, \ldots, v_t\}$ of $U/(U \cap X^\perp)$. We will first ensure that $\langle y_1, u_j\rangle = \delta_{1j}$ for all j and that $\langle x_i, v_1\rangle = \delta_{i1}$ for all i, and then continue working recursively.

There is at least one j so that $\langle y_1, v_j\rangle = 1$ (if $y_1 \perp v_j = 0$ for all j, then we'd have $y_1 \perp U$, and thus $y_1 = 0$). We swap the v_j so that $\langle y_1, v_1\rangle = 1$. We then modify both basis as follows:

$$
\begin{aligned}
y_1^{new} &= y_1, \qquad & y_i^{new} &= y_i + \langle y_i, v_1\rangle y_1 \text{ for all } i \neq 1, \\
v_1^{new} &= v_1, \qquad & v_j^{new} &= v_j + \langle y_1, v_j\rangle v_1 \text{ for all } j \neq 1.
\end{aligned}
$$

Algorithm 10: General fast Walsh transform pruned to arbitrary affine subspaces

Parameters: $L \subseteq x_0 + X \subseteq \mathbb{F}_2^n, M \subseteq u_0 + U \subseteq \mathbb{F}_2^n$, ($X, U$ subspaces).
Input: $f : L \longrightarrow \mathbb{C}$.
Output: $\widehat{f} : M \longrightarrow \mathbb{C}$.
$\mathcal{B}_X = \{y_1, \ldots, y_t\} \leftarrow \text{GetBasis}(X/(X \cap U^\perp))$;
$\mathcal{B}_U = \{v_1, \ldots, v_t\} \leftarrow \text{GetBasis}(U/(U \cap X^\perp))$;
for $k \leftarrow 1$ **to** $t-1$ **do** `// Generate orthogonal basis`
  while $\langle y_k, v_k \rangle = 0$ **do** $(v_k, \ldots, v_t) \leftarrow (v_{k+1}, \ldots, v_k)$;
  for $i \leftarrow k+1$ **to** t **do** $y_i \leftarrow y_i + \langle y_i, v_k \rangle y_k$;
  for $j \leftarrow k+1$ **to** t **do** $v_j \leftarrow v_j + \langle y_k, v_j \rangle v_k$;
end
let $g : \mathbb{F}_2^t \longrightarrow \mathbb{C},\ g(y) = 0\ \forall y \in \mathbb{F}_2^t$;
foreach $x \in L$ **do** `// Absorb the nonzero inputs`
  $(i_1, \ldots, i_t) \leftarrow \text{GetCoordinates}(\overline{x - x_0}, \mathcal{B}_X)$;
  $g(i_1, \ldots, i_t) \leftarrow g(i_1, \ldots, i_t) + (-1)^{\langle x - x_0, u_0 \rangle} f(x)$;
end
$g \leftarrow \text{FWT}(g)$; `// Fast Walsh transform of size 2^t`
foreach $u \in M$ **do** `// Generate the desired outputs`
  $(j_1, \ldots, j_t) \leftarrow \text{GetCoordinates}(\overline{u - u_0}, \mathcal{B}_U)$;
  $\widehat{f}(u) \leftarrow (-1)^{\langle x_0, u \rangle} g(j_1, \ldots, j_t)$;
end
return $\widehat{f}$;

We now check that the new basis have the desired orthogonality properties:

$$\begin{aligned}
\langle y_1^{new}, v_1^{new} \rangle &= \langle y_1, v_1 \rangle &&= 1, \\
\langle y_1^{new}, v_j^{new} \rangle &= \langle y_1, v_j \rangle + \langle y_1, v_j \rangle \langle y_1, v_1 \rangle &&= 0 \quad \text{for all } j \neq 1, \\
\langle y_i^{new}, v_1^{new} \rangle &= \langle y_i, v_1 \rangle + \langle y_i, v_1 \rangle \langle y_1, v_1 \rangle &&= 0 \quad \text{for all } i \neq 1.
\end{aligned}$$

This process can be iterated on the rest of the elements until we obtain a pair of basis $\{y_1, \ldots, y_t\}$ and $\{v_1, \ldots, v_t\}$ which verify $\langle y_i, v_j \rangle = \delta_{ij}$. We can construct the isomorphisms ϕ and ψ by mapping these bases to the standard basis of $\mathbb{F}_2^t$. □

Relying on the orthogonal basis provided by the lemma, we can prove the following result:

Theorem 4.9 (General affine pruned fast Walsh transform). *Let $\widehat{f}$ be the Walsh transform of $f \in \mathbb{C}\mathbb{F}_2^n$. We are given lists $L \subseteq x_0 + X \subseteq \mathbb{F}_2^n$ and $M \subseteq u_0 + U \subseteq \mathbb{F}_2^n$, where $x_0 + X$ and $u_0 + U$ are affine subspaces, and assume $f(x) = 0$ for all $x \notin L$. Let $t = \dim\big(X/(X \cap U^\perp)\big) = \dim\big(U/(U \cap X^\perp)\big)$. There is an algorithm which computes $\widehat{f}(u)$ for all $u \in M$ which uses 2^t memory registers and at most*

$$|L| + t2^t \text{ additions and } |M| \text{ memory accesses.} \tag{4.8}$$

Proof. Let $u = u_0 + u',\ u' \in U$ be one of the desired outputs.

$$\begin{aligned}
\widehat{f}(u) &= \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u, x \rangle} f(x) = \sum_{x' \in X} (-1)^{\langle u_0 + u', x_0 + x' \rangle} f(x_0 + x') \\
&= \sum_{x' \in X} (-1)^{\langle u, x_0 \rangle + \langle u_0, x' \rangle + \langle u', x' \rangle} f(x_0 + x') \\
&= (-1)^{\langle u, x_0 \rangle} \sum_{y \in X/(X \cap U^\perp)} (-1)^{\langle \overline{u'}, y \rangle} \sum_{x' \in y} (-1)^{\langle u_0, x' \rangle} f(x_0 + x').
\end{aligned}$$

This suggests the following algorithm:

1. For each $y \in X/(X \cap U^\perp)$, compute $g(y) = \sum_{x' \in y} (-1)^{\langle u_0, x' \rangle} f(x_0 + x')$. These sums are stored in an array g of length 2^t which is ordered according to a base of $X/(X \cap U^\perp)$ constructed using Lemma 4.8. In practice, this is done by going over all the elements x of L, computing the associated $x' = x - x_0$, and adding $f(x)$ to the bin corresponding to the class of x'. The total cost of this step is at most $|L|$ additions and subtractions. We note that we do not need to store the non-zero entries of f in memory, as they can be queried individually, only once each, and in any order.

2. We apply any fast Walsh transform algorithm on this vector, which requires $t2^t$ additions and subtractions. The result is a vector $\widehat{g}$ which contains, for each $v \in V \in U/(U \cap X^\perp)$:

$$\widehat{g}(v) = \sum_{y \in X/(X \cap U^\perp)} (-1)^{\langle v, y \rangle} \sum_{x' \in y} (-1)^{\langle u_0, x' \rangle} f(x_0 + x').$$

   The vector $\widehat{g}$ can be accessed according the basis of $U/(U \cap X^\perp)$ obtained using Lemma 4.8.

3. For each desired output $u \in M$, we separate $u = u_0 + u'$. We then access the entry of $\widehat{g}$ indexed under the class of u'. This value is then sign-swapped according to $\langle x_0, u \rangle$, and we obtain $\widehat{f}(u)$. The cost for the full list of outputs is at most $|M|$. We note that, as with the first step, each output can be queried individually and in any order. This means, for example, that we can store the vector $\widehat{g}$, which only requires 2^t registers, and query any output within $u_0 + U$ in $\mathcal{O}(1)$ afterwards. □

Example. We return to the example to illustrate how Theorem 4.9 justifies the algorithm of figure 4.6. Indeed, $U \cap X^\perp = X^\perp = \operatorname{span}\{(1,1,1,0)\}$ and $X \cap U^\perp = U^\perp = \operatorname{span}\{(1,1,0,0)\}$, which means that $t = 2$ and the transform reduces to one of size 2^2. Furthermore, the inputs and outputs of the reduced transform correspond to the bases $(\overline{(0,1,1,0)}, \overline{(0,0,0,1)})$ of $X/(X \cap U^\perp)$ and $(\overline{(0,0,1,0)}, \overline{(0,0,0,1)})$ of $U/(U \cap X^\perp)$.

We also provide the following optimality result, which states that any affine pruned algorithm must require at least as many operations as a Walsh transform of size 2^t:

Proposition 4.10. *Any algorithm which solves a given instance of Problem 4.2 in the case $L = x_0 + X$ and $M = u_0 + U$ for all compatible input vectors f must perform at least*

$$\max\left\{2^{\dim(X)}, 2^{\dim(U)}, \text{time\_walsh}\left(\dim\left(\frac{X}{X \cap U^\perp}\right)\right)\right\} \tag{4.9}$$

operations, where time_walsh(m) *is the optimal number of operations required to compute a full Walsh transform of size 2^m.*

Proof. Since any algorithm which solves the problem for any set of inputs must at least read all the active input positions, the complexity must be at least $2^{\dim(X)}$ operations. A similar argument on the outputs provides the $2^{\dim(U)}$ bound.

For the final bound, let $t = \dim\big(X/(X \cap U^\perp)\big)$, and let ϕ and ψ be the isomorphisms defined in Lemma 4.8. It suffices to reduce the computation of the Walsh transform of length 2^t of a vector $g \in \mathbb{F}_2^t$ to the affine pruning problem instance. We construct a vector $f \in x_0 + X$ as follows: for each $y \in X/(X \cap U^\perp)$, we take exactly one representative $x' \in y$ and make $f(x_0 + x') = (-1)^{\langle u_0, x' \rangle} g(\phi(y))$. All the other positions are set to zero.

When we compute the Walsh transform of f at a position $u = u_0 + u'$ using the pruned algorithm, we obtain

$$\begin{aligned}
\widehat{f}(u_0 + u') &= \sum_{x' \in X} (-1)^{\langle u_0 + u', x_0 + x' \rangle} f(x_0 + x') \\
&= (-1)^{\langle u, x_0 \rangle} \sum_{y \in X/(X \cap U^\perp)} (-1)^{\langle \overline{u'}, y \rangle + \langle u_0, x'(y) \rangle} f(x_0 + x'(y)) \\
&= (-1)^{\langle u, x_0 \rangle} \sum_{y \in X/(X \cap U^\perp)} (-1)^{\langle \overline{u'}, y \rangle} g(y) = (-1)^{\langle u_0, x_0 \rangle} \widehat{g}(\overline{u'})
\end{aligned}$$

noting that for each $y \in X/(X \cap U^\perp)$ there is a single $x'(y)$ for which $f(x_0 + x'(y)) \neq 0$.

The formula shows that, given $\psi(v) \in \mathbb{F}_2^t$, we can take any representative $u' \in v$, query $\widehat{f}(u_0 + u')$ to the pruned algorithm and multiply by $(-1)^{\langle u_0 + u', x_0 \rangle}$ in order to obtain $\widehat{g}(\psi(v))$. □

4.3 The Multi-affine Subset Case

In the previous section, we have described what we consider to be a satisfactory solution to the pruning problem when the inputs, the outputs, or both are restricted to subsets of affine subspaces of small dimension. We have also shown that the time complexity doesn't just depend on the dimensions of these subspaces, but also on their orthogonality, and have found that our solution is essentially optimal. The

next natural step is to look into algorithms for subsets of $\mathbb{F}_2^n$ which may not necessarily lie on affine subspaces of small dimension.

A first approach to tackle the unstructured pruning problem would be to find the smallest affine subspaces which cover all active inputs and all outputs, that is, to choose random $x_0 \in L$ and $u_0 \in M$, and pick $X = \text{span}(\{x - x_0\}_{x \in L})$ and $U = \text{span}(\{u - u_0\}_{u \in M})$. However, if $|L|, |M| \gg n$ it is very likely that $X = U = \mathbb{F}_2^n$, and we just obtain the traditional fast Walsh transform algorithm. In other words, this approach will only be effective for very small subsets, to the point that computing the transform point by point may even be more efficient.

However, there is one case in which the affine pruning algorithms can be very useful. In the applications of the next chapter, the lists L and M span the whole $\mathbb{F}_2^n$. However, they can be written as the union of small collections of sublists which are contained in affine subspaces of small dimension. This means we can use the linearity of the Walsh transform to separate the calculation into several parts to which Algorithm 10 can be applied.

We assume that we separate the lists L and M as disjoint unions $L = L_1 \cup \cdots \cup L_p$ and $M = M_1 \cup \cdots \cup M_q$. Let's also assume that there exist $x_0^1, \ldots, x_0^p$ and $u_0^1, \ldots, u_0^q$, as well as $X_1, \ldots, X_p$ and $U_1, \ldots, U_q$ so that $L_i \subseteq x_0^i + X_i$ for all $i = 1, \ldots, p$ and $M_j \subseteq u_0^j + U_j$ for all $j = 1, \ldots, q$. Please note that although the list families $\{L_i\}$ and $\{M_j\}$ are disjoint, the affine subspace families $\{x_0^i + X_i\}$ and $\{u_0^j + U_j\}$ are not necessarily disjoint.

We can compute the pruned Walsh transform for each pair (L_i, M_j) separately using Algorithm 10. Given f, we will denote by f_i the function equal to f but with its support restricted to L_i. We will denote by $\widehat{g_{ij}}$ the output of the internal transform used in Algorithm 10 for the lists L_i and M_j. We can then use the linearity of the Walsh transform to recombine the information from these partial transforms. Indeed, given $u \in M_j$, we can compute $\widehat{f}(u)$ as

$$\widehat{f}(u) = \sum_{i=1}^{p} \widehat{f_i}(u) = \sum_{i=1}^{p} (-1)^{\langle u, x_0^i \rangle} \widehat{g_{ij}} \left(\overline{u - u_0^j} \right). \tag{4.10}$$

Let $t_{ij} := \dim\left(X_i / (X_i \cap U_j^\perp)\right)$. The time complexity of the resulting algorithm is:

$$q|L| + \sum_{i=1}^{p} \sum_{j=1}^{q} t_{ij} \cdot 2^{t_{ij}} + p|M| \text{ additions/subtractions.} \tag{4.11}$$

The memory requirement is $\max_{j=1}^{q} \{\sum_{i=1}^{p} 2^{t_{ij}}\} + |M|$.

4.4 Conclusion and Open Problems

Over the course of this chapter, we have described a solution to the affine Walsh transform pruning problem in which it is known that the non-zero inputs lie within an affine subspace $x_0 + X$ and the desired outputs lie within an affine subspace $u_0 + U$. We have found that the time complexity is not just dependent on the dimensions of X, U, or even $X \cap U$, but that the orthogonality of these spaces plays an essential role. More specifically, we find that a pruned fast Walsh transform algorithm consisting of a reduced Walsh transform of length 2^t, where $t = \dim\left(X/(X \cap U^\perp)\right) = \dim\left(U/(U \cap X^\perp)\right)$ exists. Furthermore, we find that the calculation of the Walsh transform of length 2^t can be reduced to any affine pruning problem of arbitrary length 2^n, $n > t$, as long as $\dim\left(X/(X \cap U^\perp)\right) = t$ holds. This means that our solution to the problem is not just effective, but also efficient.

We have also shown that this algorithm can also be used when the active inputs and outputs lie on the union of some affine subspaces of small dimension. This case is of special interest to linear cryptanalysis, as will be highlighted in the next chapter. However, the algorithm for multi-affine sets considers several simplifications which may negatively impact the final complexity. In particular, we consider the subspaces X_i and U_j to be arbitrary, and that the calculations performed for each pair of subspaces are independent from each other. It is unclear whether a closer examination of the structure of these subspaces may lead to potential time complexity gains, but in any case it seems like a problem worth exploring.

It seems that, since the algorithms discussed in this chapter make heavy use of the affine space structure of the active input and output lists, such large time complexity gains may not be possible for the unstructured pruning problem. However, there are other structured pruning problems which may be of interest to cryptologists and for which significant gains may be possible. One such case is that of computing the outputs whose index has a Hamming weight bound by a small constant w. When empirically computing good output linear masks for a given input mask using the Walsh transform such

as in Section 3.3, in many cases it would seem sensible to limit the search to masks whose Hamming weight is not too large. This is a situation in which an algorithm which prunes the Walsh transform according to the Hamming weight of the indices may prove very useful.

Chapter 5

Fast Key Recovery Linear Attacks

In this chapter we present the work on FFT-accelerated linear attacks which I published with María Naya-Plasencia [FN20] and in the single author paper [Fló22]. The FFT or Walsh transform key recovery linear attack algorithm was first introduced by Collard, Standaert and Quisquater [CSQ07b]. Despite being a somewhat popular technique which can greatly reduce the time complexity of some attacks, there have been few attempts to generalise, extend or improve this approach. Our work provides a description of the algorithm which applies to more general situations than the original version, as well as a comprehensive set of tools to exploit specific aspects of the attack and the cipher in question by means of the pruned Walsh transform of Chapter 4. We also provide some application examples with specific linear cryptanalysis of several reduced-round ciphers.

Contents

Section 5.1 discusses the original contribution of [CSQ07b] and its underlying principle in detail. Section 5.2 introduces some generalisations of the algorithm which can be applied to attacks which feature key recovery over multiple rounds, as well as a short discussion of the case of multiple linear attacks. These improvements have in common that there are no modifications to the Walsh transform algorithm itself, only to the attack algorithm which makes calls to it. In Section 5.3 we discuss how to apply the Walsh transform pruning techniques from Chapter 4 in order to remove unnecessary calculations from the attack. This pruning is informed by the specific nature of the attack and the cipher: for example, we may exploit the structure of the round function, the regularity of the data, the key schedule

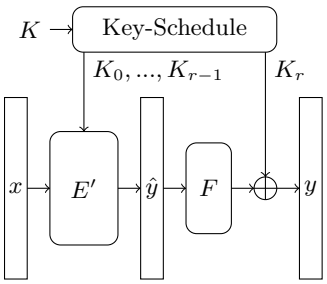


Figure 5.1: A key recovery attack on the last round of a cipher.

of the cipher, or some combination of these. Then, concrete applications to reduced-round variants of PRESENT [BKL+07], the DES [DES77] and NOEKEON [DPVAR00] are presented in Section 5.4. Finally, the chapter is rounded off with a discussion of some open problems and directions for future research.

5.1 Original Approach

We will first describe the FFT-accelerated version of Matsui's Algorithm 2 as first introduced by Collard, Standaert and Quisquater in [CSQ07b], and which is the starting point of the work showcased in this chapter. This original formulation of the algorithm applies to last-round attacks adhering to the structure shown in Figure 5.1.

We consider a last-round key recovery attack similar to the one described in Subsection 2.1.2, on an n-bit key-alternating block cipher E. We separate the last round F from the block cipher as follows: $E_K = \mathrm{ARK}_{K_r} \circ F \circ E'_K$. We consider a biased linear approximation $\langle \alpha, x \rangle + \langle \beta, \widehat{y} \rangle$ of the first $r-1$ rounds with correlation c, where $\widehat{y}$ is the state before the last round.

We suppose that computing $\langle \beta, F^{-1}(y \oplus K_r) \rangle$ from y only requires guessing $|k| < |K_r| = n$ bits of K_r, which can be selected using the mask χ. In other words, the part of K_r which has an influence on the value of the linear approximation is $k = K_r|_\chi$. We can substitute the term associated to the partial decryption of the last round for a compact map $f : \mathbb{F}_2^{|k|} \longrightarrow \mathbb{F}_2$:

$$f\left(y|_\chi \oplus K_r|_\chi\right) = \langle \beta, F^{-1}(y \oplus K_r) \rangle \text{ for all } y \in \mathbb{F}_2^n,\ K_r \in \mathbb{F}_2^n. \tag{5.1}$$

Given a collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs, we want to use Matsui's Algorithm 2 in order to recover the partial subkey k. We recall that the complexity of the algorithm is $N2^{|k|}$ one-round decryptions and $2^{|k|}$ memory registers of up to $\log N$ bits in the version from [Mat93] (Algorithm 4). If we use the algorithm with separate distillation and analysis of [Mat94a] (Algorithm 5), the time complexity becomes $\mathcal{O}(N) + \mathcal{O}\left(2^{2|k|}\right)$ instead, which results in a smaller cost when $2^{|k|} < N$.

Let us examine the calculations in more detail. Instead of the counters T_k, we know that we can equivalently compute the experimental correlation counters $\widehat{\mathrm{cor}}(k)$, which are defined as

$$N\widehat{\mathrm{cor}}(k) = \left|(x,y) \in \mathcal{D} : \langle \alpha, x \rangle + f(y|_\chi \oplus k) = 0\right| - \left|(x,y) \in \mathcal{D} : \langle \alpha, x \rangle + f(y|_\chi \oplus k) = 1\right|. \tag{5.2}$$

Each experimental correlation can be rewritten as a sum

$$N\widehat{\mathrm{cor}}(k) = \sum_{(x,y)\in\mathcal{D}} (-1)^{\langle \alpha, x \rangle + f(y|_\chi \oplus k)} = \sum_{j \in \mathbb{F}_2^{|k|}} (-1)^{f(j \oplus k)} \sum_{\substack{(x,y)\in\mathcal{D} \\ y|_\chi = j}} (-1)^{\langle \alpha, x \rangle}. \tag{5.3}$$

In the last expression of the equality, the counter j represents the relevant $|k|$ bits of the ciphertext which are xored to k. This suggests that the attack should begin by computing a vector of integer values $\mathbf{a} \in \mathbb{CF}_2^{|k|}$ whose coordinates are

$$a_j = \sum_{\substack{(x,y)\in\mathcal{D} \\ y|_\chi = j}} (-1)^{\langle \alpha, x \rangle}, \quad j \in \mathbb{F}_2^{|k|}. \tag{5.4}$$

This will constitute the distillation phase of the algorithm of [CSQ07b]. The authors also define a matrix of 1s and -1s $C \in \mathrm{GL}\left(\mathbb{CF}_2^{|k|}, \mathbb{CF}_2^{|k|}\right)$ with entries

$$c_{jk} = (-1)^{f(j \oplus k)}, \quad i, j \in \mathbb{F}_2^{|k|}. \tag{5.5}$$

The vector $N\widehat{\mathrm{cor}} = \left(N\widehat{\mathrm{cor}}(k)\right)_{k\in\mathbb{F}_2^{|k|}}$ can thus be calculated as the matrix-vector product

$$N\widehat{\mathrm{cor}} = \mathbf{a}^T C \tag{5.6}$$

However, the time complexity of constructing C and computing the matrix-vector product is still $O\left(2^{2|k|}\right)$, which is the same as with the method of [Mat94a]. However, the matrix C is *block-circulant*. In particular, this means that a fast multiplication algorithm exists, as justified by the following result, which is a consequence of the convolution theorem:

Proposition 5.1. *Let $\phi : \mathbb{F}_2^m \longrightarrow \mathbb{C}$ be any complex-valued function over $\mathbb{F}_2^m$. We consider a matrix $C \in \mathrm{GL}\left(\mathbb{CF}_2^m, \mathbb{CF}_2^m\right)$ whose entries are of the form*

$$c_{ij} = \phi(i \oplus j), \quad i, j \in \mathbb{F}_2^m. \tag{5.7}$$

Such a matrix diagonalizes as

$$2^m C = H_{2^m} \Delta H_{2^m}, \tag{5.8}$$

where H_{2^m} is the Hadamard-Sylvester matrix of size 2^m and $\Delta = \mathrm{diag}(\boldsymbol{\lambda}) \in \mathrm{GL}\left(\mathbb{CF}_2^m, \mathbb{CF}_2^m\right)$ is a diagonal matrix. The eigenvalue vector $\boldsymbol{\lambda}$ can be calculated as the matrix-vector product $H_{2^m} C_{\cdot 1}$, where $C_{\cdot 1}$ denotes the first column of C.

Proof. We will show that the columns of the Walsh-Hadamard matrix (often called Walsh functions) are eigenvectors of C. In the process, we will also obtain an expression for the associated eigenvalues. Let $h_{\cdot j}$ denote the j-th column of H_{2^m}. Its coordinates are therefore $(h_{\cdot j})_i = h_{ij} = (-1)^{\langle i,j\rangle}$. The i-th component of the matrix-vector product $Ch_{\cdot j}$ is equal to:

$$\sum_{k\in\mathbb{F}_2^m} c_{ik} h_{kj} = \sum_{k\in\mathbb{F}_2^m} \phi(i \oplus k)(-1)^{\langle k,j\rangle}.$$

Since the sum is taken over all the values of $k \in \mathbb{F}_2^m$, we can exchange the indices $i \oplus k$ for k:

$$\sum_{k\in\mathbb{F}_2^m} \phi(k)(-1)^{\langle i\oplus k,j\rangle} = \left(\sum_{k\in\mathbb{F}_2^m} \phi(k)(-1)^{\langle k,j\rangle}\right)(-1)^{\langle i,j\rangle},$$

which is a multiple of the i-th component of $h_{\cdot j}$. Furthermore, the associated eigenvalue is

$$\lambda_j = \sum_{k\in\mathbb{F}_2^m} \phi(k)(-1)^{\langle k,j\rangle},$$

which is indeed the j-th component of the product of H_{2^m} and the first column of C.

We can also see the fast multiplication formula as a consequence of the convolution theorem (see Theorem 3.6). Indeed, in order to compute the product of a vector $\mathbf{v} \in \mathbb{CF}_2^m$ by the matrix C, we must compute the entries

$$\sum_{k\in\mathbb{F}_2^m} \phi(i \oplus k)\mathbf{v}(k),$$

which are the coordinates of the convolution $\phi * \mathbf{v}$. This convolution can be obtained by applying the Walsh transform on ϕ (thus obtaining $\boldsymbol{\lambda}$) and $\mathbf{v}$, multiplying component-wise and applying the inverse Walsh transform, which is the stated decomposition of C. □

By taking $\phi(i) = (-1)^{f(i)}$ and applying this result to the matrix C, the matrix-vector product $\mathbf{a}^T C$ can then be further decomposed into:

$$2^{|k|} N\widehat{\mathrm{cor}} = \mathbf{a}^T H_{2^{|k|}} \mathrm{diag}(H_{2^{|k|}} C_{\cdot 1}) H_{2^{|k|}} \tag{5.9}$$

This decomposition of C provides the justification of Algorithm 11, which reduces computing $\widehat{\mathrm{cor}}$ to three products of the form $H_{2^{|k|}}\boldsymbol{v}$, which can in turn be evaluated efficiently with any Fast Walsh Transform algorithm with $|k|2^{|k|}$ additions and subtractions. In order to express the time complexity of the attack accurately, we will denote by ρ_D the cost of checking a plaintext-ciphertext pair in the distillation phase, by ρ_f the cost of evaluating $f(j)$, and by ρ_A and ρ_M the costs of adding and multiplying two n-bit integers, respectively.

Algorithm 11: The algorithm of [CSQ07b]

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs.
Output: The experimental correlations $\widehat{cor}(k)$ (multiplied by $2^{|k|}$).

```
// DISTILLATION PHASE
```
$\mathbf{a} \leftarrow \mathbf{0}$;
foreach $(x, y) \in \mathcal{D}$ **do**
  if $\langle \alpha, x \rangle = 0$ **then** $a_{y|_\chi} \leftarrow a_{y|_\chi} + 1$ **else** $a_{y|_\chi} \leftarrow a_{y|_\chi} - 1$;
end
```
// ANALYSIS PHASE
```
foreach $j \in \mathbb{F}_2^{|k|}$ **do** $\lambda_j \leftarrow (-1)^{f(j)}$; `// First column of` C
$\boldsymbol{\lambda} \leftarrow \mathrm{FWT}(\boldsymbol{\lambda})$; `// Eigenvalues of` C
$\mathbf{a} \leftarrow \mathrm{FWT}(\mathbf{a})$; `// Apply the FWT to a`
foreach $j \in \mathbb{F}_2^{|k|}$ **do** $a_j \leftarrow a_j \cdot \lambda_j$; `// Multiply a by` $\boldsymbol{\lambda}$
$\mathbf{q} \leftarrow \mathrm{FWT}(\mathbf{a})$; `// Apply the FWT to a`
return $\boldsymbol{q}$;

Proposition 5.2. *The previous algorithm computes* $\widehat{\mathrm{cor}}$ *with time complexity*

$$\underbrace{\rho_D N}_{\substack{\text{distillation}\\ \text{phase}}} + \underbrace{3\rho_A |k| 2^{|k|} + (\rho_f + \rho_M) 2^{|k|}}_{\text{analysis phase}} \tag{5.10}$$

The memory requirement is $2 \cdot 2^{|k|} \cdot (n + |k|)$ *bits.*

Please note that most linear attacks, in addition to the previous complexity, will also have to complete the attack by separating the largest values of $\widehat{\mathrm{cor}}(k)$ at a cost of $\rho_C 2^{|k|}$ and find the remainder of the key by exhaustive search at a cost of $\rho_E 2^{\kappa - a}$, where ρ_C is the cost of an integer comparison, ρ_E is the cost of an encryption, κ is the full size of the key, and a is the key recovery advantage as defined in Section 2.4.

The algorithm, which was first introduced in [CSQ07b], mainly considers the simplified scenario in which only the last round of a key-alternating cipher has been removed (though an application for a first and last round key-recovery attack is also sketched, and an alternative version for the case in which the key addition is performed modulo 2^n is also provided). There have been some attempts at generalisations as well as several publications in which the technique is applied to attacks on specific constructions. In [NWW11] a variant for multidimensional attacks with a fixed input mask is proposed. One of the most interesting aspects of this version is its ability to use a single distillation table for all the linear approximations in the multidimensional attack. Other technical extended versions of the algorithm have been introduced as part of dedicated cryptanalysis of specific block ciphers, for example in [ZZ15, BTV18].

However, it would seem that a significant portion of the literature on linear attacks after the publication of [CSQ07b] opts not to use this technique for a variety of reasons, with examples such as [BN13, JSZW09], where it isn't even mentioned, and [Cho10], where it is only considered as a possibility for future improvement. Some contributions apply the technique as a black box strictly in the last-round case, like [BHV14, CSQ08, BDD+15]. Finally, others assume that the same formulas can be applied directly in the multiple-round case [ER08, LC14, KKHS08, BP18].

This brief examination of the available literature highlights that the use of this technique is highly uneven, and no consensus exists on when and how it can be applied. For this reason, we thought that a specific description of a multi-round version of the attack algorithm was necessary. There were also other possible improvements which had not been discussed, such as potential ways to reduce the time complexity by exploiting relationships induced by the key schedule. We discussed these problems in [FN20, Fló22].

5.2 "External" Improvements without Pruning

In this section, we will discuss improvements to the attack algorithm of [CSQ07b] which do not require any modifications to the fast Walsh transform algorithm itself (in other words, we assume that the Walsh transform of length 2^m is an opaque procedure with $m2^m$ complexity). The first subsection will focus on generalisations of the algorithm which can be readily applied to attack scenarios in which key recovery over multiple rounds at both the input and the output of the block cipher is considered. We first describe the matrix algorithm that we introduced as part of [FN20], and then provide a small further generalisation of the algorithm which uses arrays of arbitrary dimension. The second subsection discusses how to perform the key recovery efficiently in the case of multiple linear attacks.

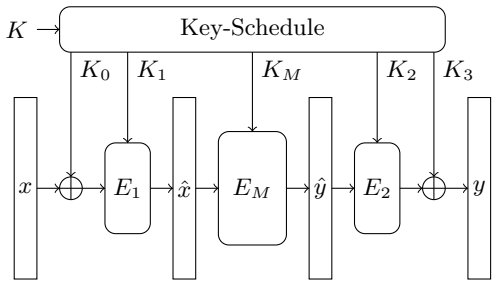


Figure 5.2: Attack schematic featuring key recovery in multiple rounds at both the plaintext and the ciphertext sides.

5.2.1 Multipart Version of the Algorithm

One of the natural extensions of the algorithm of [CSQ07b] is to perform attacks different from the last-round attack which was presented in the paper, where an outline for the first and last-round attack was also given. In [FN20], we introduced a matrix-based algorithm which successfully modelled the attack scenario corresponding to multiple rounds of key recovery at both sides of the cipher and provided a time complexity formula. However, in some instances it may be interesting to separate the function which determines the value of a linear approximation from the plaintext, ciphertext and key guess into more than two parts: for this reason we have introduced a generalised multipart attack using multidimensional arrays.

5.2.1.1 Multiple-round Attack using Matrices

In order to understand the multipart version of the algorithm using a multidimensional array, we will first describe the matrix-based multiple round attack we first proposed in [FN20]. This first generalisation of [CSQ07b] considers a block cipher E of block and key sizes n and κ which can be decomposed in the way depicted in Figure 5.2.

In this decomposition, the ciphers E_1 and E_2 will represent the first and last few rounds of the cipher, over which we wish to perform the key recovery. The parts of the key which are used in these segments will be called the *inner keys* K_1 and K_2. The first and last round subkeys will conform the *outer keys* K_0 and K_3. We suppose that the inner cipher E_M has a linear approximation $\langle \alpha, \hat{x} \rangle + \langle \beta, \hat{y} \rangle$. Similarly to the single-round case, we assume that $\langle \alpha, E_1(x \oplus K_0, K_1) \rangle$ and $\langle \beta, E_2^{-1}(y \oplus K_3, K_2) \rangle$ can be obtained from parts of x and y by guessing some bits of the keys $(K_0|K_1)$ and $(K_2|K_3)$, respectively.

We will denote the necessary part of the plaintext by i and the necessary part of the ciphertext by j, while the guessed parts of the subkeys will be denoted by k_0, k_1, k_2 and k_3. We can also consider selection masks $\chi_0, \chi_1, \chi_2, \chi_3$, so that

$$i = x|_{\chi_0},\ k_0 = K_0|_{\chi_0},\ k_1 = K_1|_{\chi_1},\ k_2 = K_2|_{\chi_2},\ j = y|_{\chi_3},\ k_3 = K_3|_{\chi_3}. \tag{5.11}$$

Let $f_1 : \mathbb{F}_2^{|k_0|} \times \mathbb{F}_2^{|k_1|} \longrightarrow \mathbb{F}_2$ and $f_2 : \mathbb{F}_2^{|k_3|} \times \mathbb{F}_2^{|k_2|} \longrightarrow \mathbb{F}_2$ be truncated maps for which

$$f_1\left(x|_{\chi_0} \oplus K_0|_{\chi_0}, K_1|_{\chi_1}\right) = \langle \alpha, E_1\left(x \oplus K_0, K_1\right) \rangle \text{ for all } x, K_0, K_1, \tag{5.12}$$

$$f_2\left(y|_{\chi_3} \oplus K_3|_{\chi_3}, K_2|_{\chi_2}\right) = \langle \beta, E_2^{-1}\left(y \oplus K_3, K_2\right) \rangle \text{ for all } y, K_3, K_2. \tag{5.13}$$

The attacker has access to a set $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs which have been generated using some unknown fixed key K. In order to perform the linear attack, they must compute, for each possible guess of the subkeys:

$$\begin{aligned} N\widehat{\mathrm{cor}}(k_0, k_1, k_2, k_3) = &\left|(x, y) \in \mathcal{D} : f_1(i \oplus k_0, k_1) \oplus f_2(j \oplus k_3, k_2) = 0\right| \\ &- \left|(x, y) \in \mathcal{D} : f_1(i \oplus k_0, k_1) \oplus f_2(j \oplus k_3, k_2) = 1\right|. \end{aligned} \tag{5.14}$$

The attack begins with a distillation phase in which a matrix $A \in \mathrm{GL}\left(\mathbb{CF}_2^{|k_0|}, \mathbb{CF}_2^{|k_3|}\right)$ is constructed from the data by examining each pair one by one. Its entries are

$$a_{ij} = \left|(x, y) \in \mathcal{D} :\ x|_{\chi_0} = i,\ y|_{\chi_3} = j\right|. \tag{5.15}$$

Similarly to [CSQ07b], we can rewrite the experimental correlation for any key guess as a sum over the entries of A:

$$N\widehat{\mathrm{cor}}(k_0,k_1,k_2,k_3) = \sum_{i\in\mathbb{F}_2^{|k_0|}} \sum_{j\in\mathbb{F}_2^{|k_3|}} a_{ij}(-1)^{f_1(i\oplus k_0,k_1)}(-1)^{f_2(j\oplus k_3,k_2)}. \tag{5.16}$$

Let us now momentarily consider that the values of k_1 and k_2 are fixed. The associated experimental correlations form a matrix $Q^{k_1,k_2} \in \mathrm{GL}\left(\mathbb{CF}_2^{|k_0|}, \mathbb{CF}_2^{|k_3|}\right)$ with entries

$$q^{k_1,k_2}_{k_0,k_3} = N\widehat{\mathrm{cor}}(k_0,k_1,k_2,k_3) \tag{5.17}$$

We can see that $Q^{k_1,k_2} = B^{k_1}AC^{k_2}$, where B^{k_1} and C^{k_2} are matrices $B^{k_1} \in \mathrm{GL}\left(\mathbb{CF}_2^{|k_0|}, \mathbb{CF}_2^{|k_0|}\right)$ and $C^{k_2} \in \mathrm{GL}\left(\mathbb{CF}_2^{|k_3|}, \mathbb{CF}_2^{|k_3|}\right)$ which are defined as

$$b^{k_1}_{k_0,i} = (-1)^{f_1(i\oplus k_0,k_1)}, \quad c^{k_2}_{j,k_3} = (-1)^{f_2(j\oplus k_3,k_2)}. \tag{5.18}$$

The matrices B^{k_1}, C^{k_2} adhere to the structure described in Proposition 5.1, which means we can use the fast Walsh transform multiplication algorithm in order to perform matrix-vector multiplication with them:

$$2^{|k_0|}B^{k_1} = H_{2^{|k_0|}}\mathrm{diag}\left(\boldsymbol{\lambda}_1^{k_1}\right)H_{2^{|k_0|}}, \text{ where } \boldsymbol{\lambda}_1^{k_1} = H_{2^{|k_0|}}B^{k_1}_{\cdot 1}, \tag{5.19}$$

$$2^{|k_3|}C^{k_2} = H_{2^{|k_3|}}\mathrm{diag}\left(\boldsymbol{\lambda}_2^{k_2}\right)H_{2^{|k_3|}}, \text{ where } \boldsymbol{\lambda}_2^{k_2} = H_{2^{|k_3|}}C^{k_2}_{\cdot 1}. \tag{5.20}$$

The matrices $Q^{k_1k_2}$ can therefore be calculated as

$$2^{|k_0|+|k_3|}Q^{k_1k_2} = H_{2^{|k_0|}}\mathrm{diag}\left(\boldsymbol{\lambda}_1^{k_1}\right)H_{2^{|k_0|}}AH_{2^{|k_3|}}\mathrm{diag}\left(\boldsymbol{\lambda}_2^{k_2}\right)H_{2^{|k_3|}}. \tag{5.21}$$

As a result, the attack can be performed efficiently using Algorithm 12, which consists of the following steps:

1. **Distillation phase:** We construct the $2^{|k_0|} \times 2^{|k_3|}$ matrix A by initialising it to zero, looking at each plaintext-ciphertext pair $(x,y) \in \mathcal{D}$ individually, finding the associated values of $i = x|_{\chi_0}$ and $j = y|_{\chi_3}$ and incrementing the corresponding $a[i,j]$ by one. The time complexity of the distillation phase is $\rho_D N$ binary operations, where ρ_D is the cost of checking one pair. The distilled data occupies $2^{|k_0|+|k_3|}$ memory registers of up to n bits.

2. **Analysis phase:** We compute all the experimental correlations $\widehat{\mathrm{cor}}(k_0,k_1,k_2,k_3)$:

   (a) We apply the fast Walsh transform on all rows and columns of A to obtain a matrix $\widehat{A}$ (in practice we can do this on the matrix A itself, as it will not be used again). The cost of this step is
   $$\rho_A\left(|k_0|+|k_3|\right)2^{|k_0|+|k_3|}, \tag{5.22}$$
   where ρ_A/ρ_M is the cost of addition/multiplication. No additional memory is used.

   (b) We construct the eigenvalue vectors $\boldsymbol{\lambda}_1^{k_1}$ and $\boldsymbol{\lambda}_2^{k_2}$ for all k_1 and all k_2 by calculating the first column of each matrix B^{k_1} and C^{k_2} and then applying the FWT. This step can be performed offline as a precomputation, and has cost
   $$\rho_{f_1}2^{|k_0|+|k_1|} + \rho_{f_2}2^{|k_2|+|k_3|} + \rho_A\left(|k_0|2^{|k_0|+|k_1|} + |k_3|2^{|k_2|+|k_3|}\right), \tag{5.23}$$
   where ρ_{f_1} and ρ_{f_2} are the costs of evaluating f_1 and f_2. These vectors are stored in $2^{|k_0|+|k_1|} + 2^{|k_2|+|k_3|}$ registers of $\max\{|k_0|,|k_3|\}$ bits.

   (c) We compute the matrix Q^{k_1,k_2} for all the values of k_1 and k_2:

      i. We start from a copy of $\widehat{A}$. We multiply each column elementwise by $\boldsymbol{\lambda}_1^{k_1}$, and each row by $\boldsymbol{\lambda}_2^{k_2}$.

      ii. We apply the FWT on all the rows and columns to obtain Q^{k_1,k_2}.

Algorithm 12: Matrix version of the FWHT attack algorithm

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs.
Output: The experimental correlations $Q^{k_1,k_2}_{k_0,k_3} = N\widehat{\mathrm{cor}}(k_0, k_1, k_2, k_3)$.

```
// DISTILLATION PHASE
```
$A \leftarrow 0 \in \mathrm{GL}\left(\mathbb{ZF}_2^{|k_0|}, \mathbb{ZF}_2^{|k_3|}\right)$;
foreach $(x, y) \in \mathcal{D}$ **do** $a[x|_{\chi_0}, y|_{\chi_3}] \leftarrow a[x|_{\chi_0}, y|_{\chi_3}] + 1$;
```
// ANALYSIS PHASE
```
foreach $i \in \mathbb{F}_2^{|k_0|}$ **do** $A[i, \cdot] \leftarrow \mathrm{FWT}(A[i, \cdot])$; `// FWT on rows of` A
foreach $j \in \mathbb{F}_2^{|k_3|}$ **do** $A[\cdot, j] \leftarrow \mathrm{FWT}(A[\cdot, j])$; `// FWT on columns of` A
foreach $k_1 \in \mathbb{F}_2^{|k_1|}$ **do** `// Compute eigenvalues of` $B^{k_1}_{\cdot 1}$
  foreach $i \in \mathbb{F}_2^{|k_0|}$ **do** $\lambda_1^{k_1}[i] \leftarrow f_1(i, k_1)$;
  $\boldsymbol{\lambda}_1^{k_1} \leftarrow \mathrm{FWT}(\boldsymbol{\lambda}_1^{k_1})$;
end
foreach $k_2 \in \mathbb{F}_2^{|k_2|}$ **do** `// Compute eigenvalues of` $C^{k_2}_{\cdot 1}$
  foreach $j \in \mathbb{F}_2^{|k_3|}$ **do** $\lambda_2^{k_2}[j] \leftarrow f_2(j, k_2)$;
  $\boldsymbol{\lambda}_2^{k_2} \leftarrow \mathrm{FWT}(\boldsymbol{\lambda}_2^{k_2})$;
end
foreach $k_1 \in \mathbb{F}_2^{|k_1|};\ k_2 \in \mathbb{F}_2^{|k_2|}$ **do** `// Compute` Q^{k_1,k_2}
  foreach $k_0 \in \mathbb{F}_2^{|k_0|};\ k_3 \in \mathbb{F}_2^{|k_3|}$ **do** $Q^{k_1,k_2}[k_0, k_3] \leftarrow A[k_0, k_3] \cdot \lambda_1^{k_1}[k_0] \cdot \lambda_2^{k_2}[k_3]$;
  foreach $k_0 \in \mathbb{F}_2^{|k_0|}$ **do** $Q^{k_1,k_2}[k_0, \cdot] \leftarrow \mathrm{FWT}(Q^{k_1,k_2}[k_0, \cdot])$;
  foreach $k_3 \in \mathbb{F}_2^{|k_3|}$ **do** $Q^{k_1,k_2}[\cdot, k_3] \leftarrow \mathrm{FWT}(Q^{k_1,k_2}[\cdot, k_3])$;
end
return $\left\{Q^{k_1,k_2} : k_1 \in \mathbb{F}_2^{|k_1|}, k_2 \in \mathbb{F}_2^{|k_2|}\right\}$;

The total cost of these operations for all the matrices Q^{k_1,k_2} is

$$2\rho_M 2^{|k_0|+|k_1|+|k_2|+|k_3|} + \rho_A \left(|k_0| + |k_3|\right) 2^{|k_0|+|k_1|+|k_2|+|k_3|}. \tag{5.24}$$

This computation requires $2^{|k_0|+|k_3|}$ working registers of up to $n + |k_0| + |k_3|$ bits. If the experimental correlations need to be stored in full (for example if the FFT algorithm is used as a part of a multiple linear attack), then $2^{|k_0|+|k_1|+|k_2|+|k_3|}$ memory registers of n bits are required (we can divide by $2^{|k_0|+|k_3|}$). If we just need to identify the key guesses with the largest correlations, then these additional registers are not needed.

3. **Search phase:** From the results of the previous step, we can find the key guesses which exhibit the largest experimental correlations and perform exhaustive searches over the rest of the key, or use the correlation tables in a multiple or multidimensional attack.

Overall, we find that the time complexity of such an attack is always

$$\mathcal{O}(N) + \mathcal{O}\left((|k_0| + |k_3|)2^{|k_0|+|k_1|+|k_2|+|k_3|}\right) \tag{5.25}$$

It's interesting to compare the costs $\rho_D, \rho_{f_1}, \rho_{f_2}, \rho_A$ and ρ_M with the cost of a block cipher encryption ρ_E. In general, ρ_D, ρ_{f_1} and ρ_{f_2} should be negligible, as they are simple operations. For most cases, ρ_A and ρ_M should be comparable to or smaller than the cost of an encryption, though this depends on the implementations of the cipher and the operations. In all the applications shown later in the chapter, we will provide a lower bound for ρ_E and an upper bound for ρ_A and ρ_M in order to write the time complexity in terms of cipher encryptions. It should be noted that, although the second term of Equation 5.24 is asymptotically dominant for a fixed n, in many cases the actual bottleneck is the first term because of ρ_M being comparatively larger than ρ_A.

One of the advantages of this version of the algorithm is its suitability to multiple and multidimensional linear attacks. In particular, if all the approximations use the same outer masks χ_0 and χ_3, the distillation phase only needs to be performed once. This generalises the results of [NWW11]. If there is no structure to the set of approximations, then the time cost of the analysis phase is multiplied by the number of approximations M. In addition, we have to consider the cost of computing the multiple/multidimensional

linear cryptanalysis statistic Q_k from the individual experimental correlation of each linear approximation, which is

$$M(\rho_M + \rho_A)2^{|k_0|+|k_1|+|k_2|+|k_3|}. \tag{5.26}$$

If there are several approximations which share the same input mask α but differ in their output masks (or the other way around), then it is possible to reuse some partial results such as $B^{k_1}\widehat{A}$, which only need to be computed once. This can lead to a further reduction in time complexity.

5.2.1.2 Generalised Multipart Attack

The previous matrix description, which we introduced in [FN20], can be used in instances in which the linear approximation is separated into two independent parts, usually corresponding to the plaintext and the ciphertext sides of the key recovery. However, in some cases we may be able to separate the value of the linear approximations into more than two independent parts. We will now show a small generalisation of the matrix approach which uses d-dimensional arrays instead, and was included in [Fló22]. We considered calling this version of the algorithm multidimensional (because it uses a multidimensional array instead of the vector of [CSQ07b] or the matrix of [FN20]) but we thought the risk of confusion with multidimensional linear cryptanalysis was non-negligible, which is why we settled for multipart.

We first consider that the linear approximation $\langle\alpha,\widehat{x}\rangle + \langle\beta,\widehat{y}\rangle$ can be rewritten as a function of the plaintext, ciphertext and key using a function such as

$$f_0(x) \oplus \underbrace{f_1(x_1 \oplus k_1^O, k_1^I) \oplus \cdots \oplus f_d(x_d \oplus k_d^O, k_d^I)}_{F(X\oplus K^O, K^I)}. \tag{5.27}$$

Here, we use $x \in \mathcal{D}$ to denote the whole data pair (in other words, the concatenation of the plaintext and the ciphertext). The vector $(x_1|\ldots|x_d) = X$ consists of the separate parts of the plaintext-ciphertext pair x which influence the value of the linear approximation. These segments can be selected with masks $\chi_1,\ldots,\chi_d$. We assume that these parts of the plaintext/ciphertext are xored to some *outer* key material $(k_1^O|\ldots|k_d^O) = K^O$. Some *inner* key material $(k_1^I|\ldots|k_d^I) = K^I$ may also be present. Note that we have $|x_i| = |k_i^O|$.

We will call the full function describing the value of the linear approximation the *key recovery map*. It consists of two parts: f_0 denotes a component of the approximation which can be calculated completely independently from the key. For example, in the original algorithm of [CSQ07b], this term would correspond to $\langle\alpha,x\rangle$. More generally, any plaintext and ciphertext material which appears linearly in the linear approximation can be treated this way. The second part F is influenced by the key. We consider that it can be divided into d independent parts. For example, in the matrix algorithm, these two parts correspond to the plaintext side key recovery and the ciphertext side key recovery. In this case the inner key K^I corresponds to k_1 and k_2, and the outer key K^O corresponds to k_0 and k_3.

Our aim is to compute all values of the experimental correlation $\widehat{\mathrm{cor}}(K^O,K^I)$:

$$\begin{aligned}
N\cdot\widehat{\mathrm{cor}}(K^O,K^I) &= \sum_{x\in\mathcal{D}}(-1)^{f_0(x)\oplus f_1(x|_{\chi_1}\oplus k_1^O,k_1^I)\oplus\cdots\oplus f_d(x|_{\chi_d}\oplus k_d^O,k_d^I)} \\
&= \sum_{x_1\in\mathbb{F}_2^{|x_1|}}\cdots\sum_{x_d\in\mathbb{F}_2^{|x_d|}}(-1)^{F(X\oplus K^O,K^I)}\underbrace{\sum_{\substack{x\in\mathcal{D}\\ x|_{\chi_1}=x_1,\ldots,x|_{\chi_d}=x_d}}(-1)^{f_0(x)}}_{A[X]} \\
&= \frac{1}{2^{|X|}}\sum_{Y\in\mathbb{F}_2^{|X|}}(-1)^{\langle K^O,Y\rangle}\left[\sum_{Z\in\mathbb{F}_2^{|X|}}(-1)^{\langle Y,Z\rangle}(-1)^{F(Z,K^I)}\right]\sum_{X\in\mathbb{F}_2^{|X|}}(-1)^{\langle Y,X\rangle}A[X] \\
&= \frac{1}{2^{|X|}}\sum_{Y\in\mathbb{F}_2^{|X|}}(-1)^{\langle K^O,Y\rangle}\left(\prod_{i=0}^{d}\widehat{f}_i(y_i,k_i^I)\right)\widehat{A}[Y],
\end{aligned} \tag{5.28}$$

using the convolution theorem, and where $\widehat{f}_i(\cdot,k_i^I)$ denotes the Walsh transform of $f_i(\cdot,k_i^I)$ for a fixed value of k_i^I. Thanks to this expression the attack can be performed as follows:

1. **Distillation phase:** We construct the $2^{|x_1|}\times\cdots\times 2^{|x_d|}$-dimensional array A from the data. This is done by looking at each plaintext-ciphertext pair x individually, computing $X = (x|_{\chi_0},\ldots,x|_{\chi_d})$, and incrementing or decrementing $A[X]$ according to $f_0(x)$. The cost of this phase is $\rho_D N$, and the memory requirement is $2^{|X|}$ registers of n bits.

Algorithm 13: Multipart version of the FWHT attack algorithm

Input: A collection $\mathcal{D} = \{(x, y = E_K(x))\}$ of N plaintext-ciphertext pairs.
Output: The experimental correlations $N \cdot \widehat{\mathrm{cor}}[K^O, K^I]$.
// DISTILLATION PHASE
$A \leftarrow 0$ array of dimension $2^{|x_1|} \times \cdots \times 2^{|x_d|}$;
foreach $x \in \mathcal{D}$ **do** $a[x|_{\chi_1}, \ldots, x|_{\chi_d}] \leftarrow A[x|_{\chi_1}, \ldots, x|_{\chi_d}] + (-1)^{f_0(x)}$;
// ANALYSIS PHASE
$A \leftarrow \mathrm{FWT}(A)$;
for $i \leftarrow 1$ **to** d **do**
  foreach $k_i^I \in \mathbb{F}_2^{|k_i^I|}$ **do** Compute $\widehat{f}_i(\cdot, k_i^I)$;
end
foreach $K^I \in \mathbb{F}_2^{|K^I|}$ **do**
  $\widehat{\mathrm{cor}}[\cdot, K^I] \leftarrow 0$ array of dimension $2^{|x_1|} \times \cdots \times 2^{|x_d|}$;
  foreach $Y \in \mathbb{F}_2^{|Y|}$ **do** $Q[Y, K^I] \leftarrow Q[Y, K^I] \cdot \prod_{i=1}^{d} \widehat{f}_i(y_i, k_i^I)$;
  $\widehat{\mathrm{cor}}[\cdot, K^I] \leftarrow \mathrm{FWT}(\widehat{\mathrm{cor}}[\cdot, K^I])/N$;
end
return $\widehat{\mathrm{cor}}$;

2. **Analysis phase:**

   (a) We apply the fast Walsh transform on the array A to obtain an array $\widehat{A}$. We note that performing the Walsh transform on any vectorized form of the array is the same as performing the Walsh transform over all dimensions. The cost of this step is thus $\rho_A |X| 2^{|X|}$, which is the cost for the full-size transform.

   (b) For each of the d functions f_i, we precompute $2^{|k_i^I|}$ tables of size $2^{|k_i^O|}$ which contain the Walsh spectrum $\widehat{f}_i(\,\cdot\,, k_i^I)$. This step can be precomputed and its cost is $\rho_A \sum_{i=1}^{d} |k_i^O| 2^{|k_i^I|+|k_i^O|}$. The memory requirement is $\sum_{i=1}^{d} 2^{|k_i^I|+|k_i^O|}$ registers. One of the main advantages of the multipart method is the fact that the more subdivisions we perform of the key recovery map, the smaller the memory requirement will be for these precomputed tables.

   (c) For each value of the internal key guess K^I:

      i. Multiply each entry $\widehat{A}[Y]$ of $\widehat{A}$ by $\prod_{i=1}^{d} \widehat{f}_i(y_i, k_i^I)$.
      ii. Apply another FWT to obtain an array containing $N \cdot \widehat{\mathrm{cor}}[\,\cdot\,, K^I]$.

      This step is usually the bottleneck of the algorithm. Its computational cost is

$$\rho_D d 2^{|K^I|+|K^O|} + \rho_A |K^O| 2^{|K^I|+|K^O|}. \tag{5.29}$$

      In terms of memory, we require $2^{|K^O|}$ memory registers if we just wish to compute all the experimental correlations without storing them, and $2^{|K^I|+|K^O|}$ if we want the full experimental correlations array to be stored for later use.

3. **Search phase.**

This version of the algorithm provides some extra flexibility when compared to the matrix version, and it can make the resulting attacks slightly more efficient and easier to implement. Subdividing the key recovery map into as many independent parts as possible can, for example, be of great help in multiple or multidimensional attacks. If a group of approximations only vary in one part, then we can compute all the Walsh transforms and eigenvalue multiplications with respect to the other parts first. As a result, only the computations related to the different part have to be carried out separately for each approximation. We can also see that the distillation phase reuse trick can be used only if the approximations use the same function f_0.

5.2.1.3 Dependencies in the Inner Key Guess

In key recovery attacks, it is often interesting to try and identify the redundancies in the key guess which are induced by the key schedule. Given the limited size of the master key K with respect to the full round subkey sequence $(K_0, \ldots, K_r)$, it is often the case that determining a large enough part of one round subkey will also determine some bits of other round subkeys. Depending on the key recovery

algorithm, for example in the original version of Matsui's Algorithm 2 (Algorithm 4), it is sometimes trivial to incorporate these relationships into the attack. Since the main part of the attack consists of a loop over all possible key values, it suffices to perform the guessing of the key in such a way which skips all guesses which are impossible due to the key schedule. However, such optimisations become much more complicated in the case of attacks which are more algorithmically complex, such as the ones presented in this chapter.

Any dependencies between the inner key guesses $k_1^I, \ldots, k_d^I$ can be treated in the same way key dependencies are treated in Matsui's Algorithm 2. The most costly part of the analysis phase consists of a loop over all possible values of the internal key guess $(k_1^I, \ldots, k_d^I)$. Assuming that we have found a way to enumerate all possible values of this internal guess according to the key schedule efficiently, it suffices to use this reduced guess as the index for the for loop. Concerning the time complexity, we can see that this technique can be described by considering that K^I is not a simple concatenation of all the k_i^I, but a minimal guess of key material which fully determines the values of the latter. This means the complexity formulas we have established are still valid, but we have substituted the parameter $|K^I|$ for a smaller value.

The next natural step is to incorporate the outer key guess K^O into this key schedule-aware, efficient guessing game. Indeed, given a guess of K^I, we can in many cases obtain a lot of information about K^O from it. Furthermore, there may be relationships between the parts $k_1^O, \ldots, k_d^O$. However, this information is a lot more difficult to incorporate into the attack algorithm: the fact that a specific value of $(k_1^O, \ldots, k_d^O)$ is not allowed by the key schedule simply means that the corresponding output of the last set of Walsh transforms does not need to be calculated. At first glance, this should not provide much of a reduction in terms of time complexity gain, as the fast Walsh transform is an apparently indivisible algorithm which can only compute all the outputs at the same time. This was our main motivation to investigate pruning techniques for the fast Walsh transform algorithm. The resulting algorithmic improvements will be discussed in Section 5.3.

5.2.2 Partial Key Guessing for Multiple Attacks

We will now discuss specifically the case in which the FFT linear key recovery algorithm is used as part of a multiple linear attack. In particular, in this case it is often possible to make use of the key schedule of the cipher without requiring any actual pruning of the fast Walsh transform (although the best results can be obtained by combining both approaches). The main idea we will apply in order to reduce the time and memory complexity is to perform the FFT algorithm separately for each linear approximation and then combine the information to obtain the multiple linear cryptanalysis statistic Q_k using some judicious guessing of different parts of the key for each linear approximation. The following approach generalises the specific attacks of [ZZ15] and [BTV18], and was first introduced as part of [FN20].

We consider a set of M linear approximations ν_i of the inner cipher E_M which will be of the form $\nu_i : \langle \alpha_i, \hat{x} \rangle + \langle \beta_i, \hat{y} \rangle$. We also assume that each approximation has a key recovery function

$$f_0^i(x) + F^i\left(X^i \oplus K^{O,i}, K^{I,i}\right) \tag{5.30}$$

We note that the functions f_0^i and F^i as well as the plaintext and key maskings X^i, $K^{O,i}$ and $K^{I,i}$ may be different for each linear approximation. For this reason, we will also define X, K^O and K^I as the minimum parts of the plaintext and the subkeys which wholly determine the values of X^i, $K^{O,i}$ and $K^{I,i}$ for all the approximations (in other words, these are computed using extended masks which are valid for all the ν_i). We suppose that each approximation has some masks $\chi^{O,i}$ and $\chi^{I,i}$ so that $X^i = X|_{\chi^{O,i}}$, $K^{O,i} = K^O|_{\chi^{O,i}}$, and $K^{I,i} = K^I|_{\chi^{I,i}}$. Finally, we also consider that there is a global key guess k_T which determines the values of all the $K^{O,i}$ and $K^{I,i}$, even if it is non-linearly. We will also suppose that the pairs $(f_0^i, \chi^{O,i})$ and $(\chi^{O,i}, \chi^{I,i})$ take M_1 and M_2 different values over the set of M approximations, respectively.

In a χ^2 multiple or multidimensional attack, the attacker wishes to compute the multiple linear cryptanalysis statistic for each global key guess k_T:

$$Q(k_T) = \sum_{i=1}^{M} \widehat{\mathrm{cor}}^i(k_T)^2, \tag{5.31}$$

where $\widehat{\mathrm{cor}}^i(k_T)$ denotes the experimental correlation of approximation ν_i for the global key guess k_T. We note that in order to calculate one particular $\widehat{\mathrm{cor}}^i(k_T)$ we only require the values of $K^{O,i}$ and $K^{I,i}$, as well as the frequency of each possible value of X^i in the data sample. In this situation, the attacker can perform the following modified attack:

1. **Distillation phase:** We construct M_1 different distillation tables: one for each possible value of the pair $(f_0^i, \chi^{O,i})$ within the set of approximations. Given a pair (f_0, χ^O), the table $A^{(f_0,\chi)}$ will have dimension $2^{\mathrm{wt}(\chi)}$, and its entries will be equal to

$$A^{(f_0,\chi)}[X] = \sum_{\substack{x\in\mathcal{D} \\ x|_\chi = X}} (-1)^{f_0(x)}.$$

   We note that from this distilled table we can compute the experimental correlations of any linear approximation for which $(f_0^i, \chi^{O,i}) = (f_0, \chi^O)$.

2. **Analysis phase (I):** In the first step of the analysis phase, we compute the experimental correlations for each linear approximation separately. For each approximation ν_i, we apply the multipart FWT algorithm to compute its experimental correlation for all guesses of $K^{O,i}$ and $K^{I,i}$ from the distillation table $A^{(f_0^i,\chi^{O,i})}$. The result is a table of size $2^{|K^{O,i}|+|K^{I,i}|}$ which contains $\widehat{\mathrm{cor}}^i(K^{O,i}, K^{I,i})$ for all the key guesses $(K^{O,i}, K^{I,i})$.

3. **Analysis phase (II):** In the second step of the analysis phase, we combine the information obtained about each approximation in the previous step into the multiple linear key recovery statistic. This is achieved in two merging stages:

   (a) We merge the M tables which were obtained in the previous step into M_2 "condensed" tables by adding the square correlations of all the approximations which correspond to the same choice of subkey bits guess, that is, one table for each possible value of the pair $(\chi^{O,i}, \chi^{I,i})$. The associated condensed tables thus contain the coefficients:

$$\sum_{\substack{i,\ (\chi^{O,i},\chi^{I,i}) \\ =(\chi_1,\chi_2)}} \widehat{\mathrm{cor}}^i(K^{O,i}, K^{I,i})^2 \quad \text{for all} \quad K^{O,i}, K^{I,i} \tag{5.32}$$

   (b) For each possible guess of the partial master key k_T, we use the key schedule to compute the associated values of $K^{O,i}, K^{I,i}$. We add up the associated entries in the tables obtained in the previous step to obtain the statistic $Q(k_T)$.

4. **Search phase:** The guesse(s) of k_T which exhibit the largest values of $Q(k_T)$ in the previous step are tested using exhaustive search over the remainder of the key until either the full key K is found, or the attack fails.

The performance of this algorithm can be improved further by pruning the individual calls to the FFT linear cryptanalysis algorithm using the methods shown in the next section. We also note that steps 2 and 3(a) can be mixed in order to reduce the memory requirement, as we can compute the experimental correlations for one approximation and immediately add them to the appropriate condensed table instead of storing them separately for every approximation and combining them afterwards.

If ρ_{KS} denotes the cost of computing $(K^{O,i}, K^{I,i})$ from k_T, then the cost of this attack algorithm is, after removing some negligible terms:

$$\begin{aligned} &\underbrace{M_1\rho_D N}_{\text{distillation}} + \underbrace{\sum_{i=1}^{M} \left(\rho_M 2^{|K^{O,i}|+|K^{I,i}|} \ + \ \rho_A |K^{O,i}| 2^{|K^{O,i}|+|K^{I,i}|}\right)}_{\text{analysis I}} \\ &\quad + \underbrace{\sum_{i=1}^{M} (\rho_M + \rho_A)\, 2^{|K^{O,i}|+|K^{I,i}|}}_{\text{analysis II (a)}} \ + \ \underbrace{(\rho_{KS} + M_2\rho_A) 2^{|k_T|}}_{\text{analysis II (b)}}, \end{aligned} \tag{5.33}$$

in addition to the cost of the search phase. In terms of memory, the distillation phase requires a total of $2^{|K^{O,i}|+|K^{I,i}|}$ registers for each different value of $(f_0^i, \chi^{O,i})$. For the M_2 condensed correlation tables, we require $2^{|K^{O,i}|+|K^{I,i}|}$ positions for each different value of $(\chi^{O,i}, \chi^{I,i})$. This algorithm can produce very large time complexity gains in the case of multiple linear cryptanalysis (especially when the $|k^{O,i}|$ and $|K^{I,i}|$ are significantly smaller than $|K^O|$ and $|K^I|$). Its success is more limited in the case of multidimensional attacks, as multidimensional linear approximations, by their very definition, often guarantee that there is a linear approximation for which $|K^{O,i}|$ and $|K^{I,i}|$ are maximal.

5.3 "Internal" Improvements with Pruning

This section explores how the Walsh transform pruning techniques of the previous chapter can be used to accommodate dependencies and redundancies in specific attack scenarios into the FWT-accelerated linear key recovery attack algorithm, in order to achieve time and memory complexity gains. We will consider three aspects of the key recovery which may feature useful properties specific to the attack: the data, the key guess, and the key recovery map itself. The following is an outline of our strategy to using each one of them.

- **Sparsity in the data:** In many cases, it is possible that the distillation array A is highly sparse. In some cases, it is the cipher construction itself which guarantees that some fixed positions $A[X]$ are always zero. For example, the expansion map E of the DES makes it so some pairs of bits at the subkey addition step will always have the same value. In other cases, A is sparse simply because the number of plaintext-ciphertext pairs N is smaller than the number of registers $2^{|X|}$. These sparsity properties can be used to save memory by storing A as a more compact array of smaller dimension, or even by storing A as a list of non-zero entries. They can also be used to prune the first Walsh transform which computes $\widehat{A}$ on the input side. In fact, in many cases we can skip building the array A altogether by performing the first step of the affine pruned fast Walsh transform as part of the distillation phase.
- **Key schedule-induced dependencies:** We have already briefly discussed that, due to the key schedule, it is possible that determining the value of the inner key guess K^I will impose some restrictions on the outer key guess K^O. These can be used to reduce the memory complexity in multiple linear attacks (since the correlations for impossible key guesses are not required, they do not need to be stored). In addition, the restriction on the values of K^O can be used to prune the last set of fast Walsh transforms at the output side.
- **Structure of the Walsh spectrum of F:** In the FWT linear key recovery attack algorithm, the Walsh spectrum of the key recovery map for a fixed inner key guess $F(\cdot, K^I)$ plays an essential role, as we need to perform a pointwise multiplication by an eigenvalue vector which is calculated as its Walsh transform $\widehat{F}(\cdot, K^I)$. As a result, any information about this Walsh spectrum could potentially be used in order to avoid unnecessary computations. We note the following possibilities:
  - The presence of any zeroes in $\widehat{F}(\cdot, K^I)$ means that the associated positions in the array $\widehat{A}$ will be multiplied by zero. This means that we do not need to compute these positions of $\widehat{A}$, thus pruning the first set of fast Walsh transforms at the output side, and that the associated input for the last set of fast Walsh transforms will be equal to zero, thus pruning them at the input side. It also directly reduces the number of integer multiplications (as we do not need to actually perform multiplications by zero) and the memory requirement.
  - If we can find a compact description for the Walsh spectrum of $F(\cdot, K^I)$, for example by writing each coefficient as a product of Walsh coefficients of some smaller maps, this might be helpful when performing the eigenvalue multiplication step as efficiently as possible, as well as saving memory.
  - Finally, there are some instances in which the coefficients $\widehat{F}(\cdot, K^I)$ can be obtained from $\widehat{F}(\cdot, 0)$ at a very small cost (for example, through some sign swaps). This can help us reduce the cost of the multiplication step even further.

The identification of the first two properties often only requires simple inspection of the construction of the cipher, the attack parameters and the key schedule. However, in the case of the Walsh spectrum of the key recovery map, some extra tools may be required. Furthermore, it is possible that the map F has a small amount of zeroes, but some similar-enough maps have a very large amount of zeroes. These issues are covered in the following subsection.

5.3.1 Zeroes in the Walsh Spectra of the Key Recovery Map

This subsection adapts some known results on the Walsh transform into tools which can be used to identify zeroes in the Walsh spectra of common block cipher constructions. We focus on constructions featuring bricklayer nonlinear maps and linear transformations, such as SPNs. We will also illustrate how in some cases rejecting a carefully-chosen subset of plaintext-ciphertext pairs, thus modifying the key recovery map slightly, can drastically reduce the number of non-zero coefficients. For that purpose, we require the Walsh spectrum restricted to a subset X:

Definition 5.3. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a vectorial Boolean function, and $X \subseteq \mathbb{F}_2^n$ a subset of its domain. The Walsh transform of f restricted to X is defined as the matrix with coefficients*

$$\widehat{f_{x\in X}}(u,v) = \sum_{x\in X} (-1)^{\langle u,x\rangle \oplus \langle v,f(x)\rangle} \tag{5.34}$$

We can also define the transform restricted to a subset of the output space $Y \subseteq \mathbb{F}_2^m$ as the matrix with coefficients $\widehat{f_{f(x)\in Y}} = \widehat{f_{x\in f^{-1}(Y)}}$.

This concept should not be confused with the input and output restrictions imposed on the fast Walsh transform algorithm when pruning. We will now restate some of the results of Section 3.3 in the case of input and output space restrictions. We start with Proposition 3.15:

Proposition 5.4. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^l$ and $g : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^m$ be vectorial Boolean functions. Let $X \subseteq \mathbb{F}_2^n$, $Z \subseteq \mathbb{F}_2^l$ and $Y \subseteq \mathbb{F}_2^m$ be subsets. The Walsh transform of $g \circ f$ restricted to sets in the input, the output and the intermediate space can be calculated using the following formulas:*

$$2^l \widehat{g \circ f}_{x\in X}(u,v) = \sum_{w\in\mathbb{F}_2^l} \widehat{f_{x\in X}}(u,w) \cdot \widehat{g}(w,v) \tag{5.35}$$

$$2^l \widehat{g \circ f}_{f\circ g(x)\in Y}(u,v) = \sum_{w\in\mathbb{F}_2^l} \widehat{f}(u,w) \cdot \widehat{g_{g(z)\in Y}}(w,v) \tag{5.36}$$

$$2^l \widehat{g \circ f}_{f(x)\in Z}(u,v) = \sum_{w\in\mathbb{F}_2^l} \widehat{f_{f(x)\in Z}}(u,w)\widehat{g}(w,v) = \sum_{w\in\mathbb{F}_2^l} \widehat{f}(u,w)\widehat{g_{z\in Z}}(w,v) \tag{5.37}$$

Proof. The proofs are the same as for Proposition 3.15, and also use Lemma 3.12.

$$\begin{aligned}
\sum_{w\in\mathbb{F}_2^l} \widehat{f_{x\in X}}(u,w)\cdot\widehat{g}(w,v) &= \sum_{w\in\mathbb{F}_2^l} \left(\sum_{x\in X} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle}\right)\cdot\left(\sum_{z\in\mathbb{F}_2^l} (-1)^{\langle w,z\rangle+\langle v,g(z)\rangle}\right)\\
&= \sum_{w\in\mathbb{F}_2^l}\sum_{x\in X}\sum_{z\in\mathbb{F}_2^l} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle+\langle w,z\rangle+\langle v,g(z)\rangle}\\
&= \sum_{x\in X}\sum_{z\in\mathbb{F}_2^l} (-1)^{\langle u,x\rangle+\langle v,g(z)\rangle} \underbrace{\sum_{w\in\mathbb{F}_2^l} (-1)^{\langle w,f(x)\rangle+\langle w,z\rangle}}_{2^l \text{ if } z=f(x),\ 0 \text{ otherwise}}\\
&= 2^l \sum_{x\in X} (-1)^{\langle u,x\rangle+\langle v,g(f(x))\rangle} = 2^l \widehat{g\circ f}_{x\in X}(u,v).
\end{aligned}$$

$$\begin{aligned}
\sum_{w\in\mathbb{F}_2^l} \widehat{f}(u,w)\cdot\widehat{g_{g(z)\in Y}}(w,v) &= \sum_{w\in\mathbb{F}_2^l} \left(\sum_{x\in\mathbb{F}_2^n} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle}\right)\cdot\left(\sum_{z\in g^{-1}(Y)} (-1)^{\langle w,z\rangle+\langle v,g(z)\rangle}\right)\\
&= \sum_{w\in\mathbb{F}_2^l}\sum_{x\in\mathbb{F}_2^n}\sum_{z\in g^{-1}(Y)} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle+\langle w,z\rangle+\langle v,g(z)\rangle}\\
&= \sum_{x\in\mathbb{F}_2^n}\sum_{z\in g^{-1}(Y)} (-1)^{\langle u,x\rangle+\langle v,g(z)\rangle} \underbrace{\sum_{w\in\mathbb{F}_2^l} (-1)^{\langle w,f(x)\rangle+\langle w,z\rangle}}_{2^l \text{ if } z=f(x),\ 0 \text{ otherwise}}\\
&= 2^l \sum_{x\in(g\circ f)^{-1}(Y)} (-1)^{\langle u,x\rangle+\langle v,g(f(x))\rangle} = 2^l \widehat{g\circ f}_{g(f(x))\in Y}(u,v).
\end{aligned}$$

$$\begin{aligned}
\sum_{w\in\mathbb{F}_2^l} \widehat{f}(u,w)\cdot\widehat{g_{z\in Z}}(w,v) &= \sum_{w\in\mathbb{F}_2^l} \left(\sum_{x\in\mathbb{F}_2^n} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle}\right)\cdot\left(\sum_{z\in Z} (-1)^{\langle w,z\rangle+\langle v,g(z)\rangle}\right)\\
&= \sum_{w\in\mathbb{F}_2^l}\sum_{x\in\mathbb{F}_2^n}\sum_{z\in Z} (-1)^{\langle u,x\rangle+\langle w,f(x)\rangle+\langle w,z\rangle+\langle v,g(z)\rangle}\\
&= \sum_{x\in\mathbb{F}_2^n}\sum_{z\in Z} (-1)^{\langle u,x\rangle+\langle v,g(z)\rangle} \underbrace{\sum_{w\in\mathbb{F}_2^l} (-1)^{\langle w,f(x)\rangle+\langle w,z\rangle}}_{2^l \text{ if } z=f(x),\ 0 \text{ otherwise}}\\
&= 2^l \sum_{x\in f^{-1}(Z)} (-1)^{\langle u,x\rangle+\langle v,g(f(x))\rangle} = 2^l \widehat{g\circ f}_{f(x)\in Z}(u,v).
\end{aligned}$$

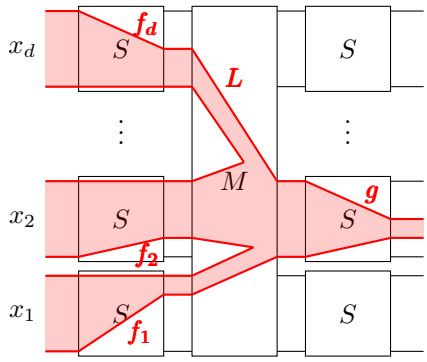


Figure 5.3: An example of how Proposition 5.5 can be used to model key recovery over two rounds of an SPN construction when a single output bit of the second Sbox layer is required.

$$\begin{aligned}\sum_{w\in\mathbb{F}_2^l} \widehat{f_{f(x)\in Z}}(u,w)\cdot\widehat{g}(w,v) &= \sum_{w\in\mathbb{F}_2^l}\left(\sum_{x\in f^{-1}(Z)}(-1)^{\langle u,x\rangle+\langle w,f(x)\rangle}\right)\cdot\left(\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle w,z\rangle+\langle v,g(z)\rangle}\right)\\ &= \sum_{w\in\mathbb{F}_2^l}\sum_{x\in f^{-1}(Z)}\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle u,x\rangle+\langle w,f(x)\rangle+\langle w,z\rangle+\langle v,g(z)\rangle}\\ &= \sum_{x\in f^{-1}(Z)}\sum_{z\in\mathbb{F}_2^l}(-1)^{\langle u,x\rangle+\langle v,g(z)\rangle}\underbrace{\sum_{w\in\mathbb{F}_2^l}(-1)^{\langle w,f(x)\rangle+\langle w,z\rangle}}_{2^l \text{ if } z=f(x),\ 0 \text{ otherwise}}\\ &= 2^l\sum_{x\in f^{-1}(Z)}(-1)^{\langle u,x\rangle+\langle v,g(f(x))\rangle} = 2^l\widehat{g\circ f}_{f(x)\in Z}(u,v).\end{aligned}$$ □

Using this result together with Propositions 3.13 and 3.14, we can often obtain compact formulas for the Walsh coefficients of some key recovery maps, such as the following:

Proposition 5.5. *Let $f_i:\mathbb{F}_2^{n_i}\longrightarrow\mathbb{F}_2^{l_i}$, $i=1,\ldots,d$ be d balanced vectorial Boolean functions, let $L:\mathbb{F}_2^{\sum_i l_i}\longrightarrow\mathbb{F}_2^l$ be a linear map, and let $g:\mathbb{F}_2^l\longrightarrow\mathbb{F}_2$ be a Boolean function. In the applications, the f_i will be some Sboxes with possibly truncated outputs, L will be a truncation of the linear layer, and g will be a linear combination of outputs of an Sbox layer. We also consider a subset of the inputs of g, $Z\subseteq\mathbb{F}_2^l$. We consider the composition $h=g\circ L\circ\mathbf{F}$, where $\mathbf{F}$ is the bricklayer function $\mathbf{F}(x_1,\ldots,x_d)=(f_1(x_1)|\ldots|f_d(x_d))$. Then the Walsh coefficients of h can be obtained through the following formula:*

$$\widehat{h}_{L(\mathbf{F}(x))\in Z}(u_1,\ldots,u_d)=\frac{1}{2^l}\underbrace{\sum_{w_1\in\mathbb{F}_2^{l_1}}\cdots\sum_{w_d\in\mathbb{F}_2^{l_d}}\sum_{v\in\mathbb{F}_2^l}}_{\substack{w_i=0 \text{ if } u_i=0\\(w_1,\ldots,w_d)=L^tv}}\prod_{i=1}^{d}\widehat{f}_i(u_i,w_i)\widehat{g_{z\in Z}}(v). \tag{5.38}$$

Proof. We first use Proposition 5.4 to write the Walsh coefficients of h as the sum

$$\widehat{h}_{L(\mathbf{F}(x))\in Z}(u_1,\ldots,u_d)=\frac{1}{2^{\sum_i l_i+l}}\sum_{w\in\mathbb{F}_2^{\sum_i l_i}}\sum_{v\in\mathbb{F}_2^l}\widehat{\mathbf{F}}(u,w)\widehat{L}(w,v)\widehat{g_{z\in Z}}(v).$$

We can simplify this formula. First, we know that $\widehat{L}(w,v)\neq 0$ if and only if $w=L^tv$, in which case $\widehat{L}(w,v)=2^{\sum_i l_i}$, according to Proposition 3.13. This means we only have to consider the sums over the w_i for which an appropriate v exists, and vice versa. Furthermore, we can write $\widehat{\boldsymbol{F}}(u,w)=\prod_{i=1}^d\widehat{f}_i(u_i,w_i)$ according to Proposition 3.14. Since the f_i are balanced, we have $\widehat{f}_i(0,w_i)=0$ if $w_i\neq 0$, so we can assume $w_i=0$ for the i for which $u_i=0$. □

As this result may seem a little abstract without context, let us formulate the specific case in which all $l_i=1$. An application example of this case is that of SPN constructions which use a bit permutation as the linear layer, because each input of the Sbox g will be an output bit of an Sbox in the previous round. However, it applies in all cases in which a single bit of information about the output of each Sbox is required.

Corollary 5.6. *Let $f_i : \mathbb{F}_2^{n_i} \longrightarrow \mathbb{F}_2,\ i = 1, \ldots, l$ be l balanced Boolean functions, and let and $g : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$ be another Boolean function. We consider the function $h(x_1, \ldots, x_l) = g(f_1(x_1)| \ldots |f_l(x_l))$ and the subset $Z \subseteq \mathbb{F}_2^d$. Then*

$$\widehat{h_{\mathbf{F}(x)\in Z}}(u_1, \ldots, u_l) = \frac{2^{\sum_{i,u_i=0} n_i}}{2^l} \widehat{g_{z\in Z}}(w(u_1, \ldots, u_l)) \prod_{i,u_i\neq 0} \widehat{f}_i(u_i), \tag{5.39}$$

where $w(u_1, \ldots, u_l)_i = \begin{cases} 0 & \text{if } u_i = 0 \\ 1 & \text{if } u_i \neq 0 \end{cases}$.

We'll show how the previous result describes $\widehat{h_{f(x)\in Z}}$ and its zeroes in a compact manner. We first look at $\widehat{g_{z\in Z}}$. Given any $w \in \mathbb{F}_2^l$ so that $\widehat{g_{z\in Z}}(w) = 0$, we can deduce that $\widehat{h_{\mathbf{F}(x)\in Z}}(u_1, \ldots, u_l) = 0$ for all $(u_1, \ldots, u_l)$ so that $w = w(u_1, \ldots, u_l)$. Furthermore, for the $(u_1, \ldots, u_l)$ for which $\widehat{g_{z\in Z}}(w(u_1, \ldots, u_l)) \neq 0$, the Walsh coefficient $\widehat{h_{\mathbf{F}(x)\in Z}}(u_1, \ldots, u_l)$ can be written as the product of $\widehat{g_{z\in Z}}(w(u_1, \ldots, u_l))$ and the $\widehat{f}_i(u_i)$ corresponding to each $u_i \neq 0$.

A very interesting situation presents itself in the case $\widehat{g_{z\in Z}}(1, \ldots, 1) = 0$. When this happens, given any $(u_1, \ldots, u_l)$ so that $u_i \neq 0$ for all i, we know that $\widehat{h_{f(x)\in Z}}(u_1, \ldots, u_l) = 0$. This means that any non-zero coefficient of this Walsh spectrum must verify that $u_i = 0$ for at least one i. As a result, the non-zero Walsh coefficients $\widehat{h_{\mathbf{F}(x)\in Z}}(u_1, \ldots, u_l)$ can be separated into l vector subspaces U_i of dimensions $\sum_j n_j - n_i,\ i = 1, \ldots, l$. Each U_i is determined by the n_i linear conditions $u_i = 0$.

In the case $\widehat{g}(1 \ldots 1) = 0$, this decomposition of the Walsh spectrum into vector subspaces of smaller dimension is obtained without any modifications to the key recovery map. However, when $\widehat{g}(1 \ldots 1) \neq 0$, we would like to be able to choose some large $Z \subseteq \mathbb{F}_2^d$ so that $\widehat{g_{z\in Z}}(1, \ldots, 1) = 0$. We can use the following result:

Proposition 5.7. *Let $g : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$ be a map for which $\widehat{g}(1 \ldots 1) = a \neq 0$. There exists $Z \subseteq \mathbb{F}_2^l$ with $|Z| = 2^l - |a|$ so that $\widehat{g_{z\in Z}}(1 \ldots 1) = 0$.*

Proof. It is sufficient to remove $|a|$ input values x whose contribution to the Walsh coefficient has the same sign as a. □

But what does this modification mean in terms of the key recovery map? We recall the definition of $\widehat{h_{\mathbf{F}(x)\in Z}}$. It is the Walsh transform of the complex function which is valued $(-1)^{h(x)}$ for all x so that $\mathbf{F}(x) \in Z$, and zero elsewhere. In terms of key recovery, this means we have substituted the key recovery map, which is normally valued ± 1 according to the value of the linear approximation, for a modified map which is zero when $\mathbf{F}(x) \notin Z$. In other words, we are rejecting the plaintext-ciphertext pairs for which the input of g is not in Z. Assuming that the correlation of the linear approximation is the same within this subset of plaintexts as it is in the whole plaintext space, we conclude that the resulting attack will have the same parameters, with the only difference that $(2^l - |Z|)/2^d$ plaintext-ciphertext pairs will be rejected for each key guess, meaning that the data sample must be increased by a factor of $2^l/|Z|$ in order to compensate.

These results have so far allowed us to describe *static* key recovery maps $F(X \oplus K^O)$ which do not feature inner key guesses. However, we must also consider the incorporation of key material within these maps $F(X \oplus K^O, K^I)$. The following result shows that, in the case in which all $l_i = 1$, the xoring of a round subkey between rounds only changes the sign of the coefficients, which in particular means that the positions of the zero coefficients remain unaltered.

Corollary 5.8. *Let $f_i : \mathbb{F}_2^{n_i} \longrightarrow \mathbb{F}_2,\ i = 1, \ldots, l$ be Boolean functions and let $g : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$ be another Boolean function, and let $k \in \mathbb{F}_2^l$ be a fixed parameter. We consider the parametric function $h(x_1, \ldots, x_l; k) = g((f_1(x_1), \ldots, f_l(x_l)) \oplus k)$ and the subset $Z \subseteq \mathbb{F}_2^l$. Then*

$$\widehat{h(\cdot, k)}_{\mathbf{F}(x)\oplus k\in Z}(u_1, \ldots, u_l) = (-1)^{\langle k, w(u_1,\ldots,u_l)\rangle} \widehat{h(\cdot, 0)}_{\mathbf{F}(x)\in Z}(u_1, \ldots, u_l). \tag{5.40}$$

5.3.2 Time Complexity Analysis

We will now provide a complexity analysis of the fast Walsh transform linear key recovery algorithm when the previously described optimisations are used. For the sake of clarity, we will assume that the target linear approximation is of the form $f_0(x) + f(X \oplus K^O, K^I)$. In practice, we can also consider multipart linear key recovery maps. We also assume that the following conditions are satisfied:

- The parts of the plaintext-ciphertext pair X which are xored with the outer key guess K^O lie in an affine subspace of the form $x_0 + Y \subseteq \mathbb{F}_2^{|K^O|}$.

- The non-zero Walsh coefficients of the key recovery map $F(\cdot, 0)$ lie in the union of l affine subspaces of the forms $u_0^i + U_i \subseteq \mathbb{F}_2^{|K^O|}$, $i = 1, \ldots, l$. We will denote the actual number of non-zero coefficients in $u_0^i + U_i$ by L_i. We also assume that the non-zero Walsh coefficients of $F(\cdot, K^I)$ occupy the same subspaces. If the latter is not true, each value of K^I must be treated separately, and the cost of the analysis phase is multiplied by $2^{|K^I|}$.

- When the key schedule of the cipher is taken into account, for a given guess of K^I, the possible values of K^O lie within an affine subspace of the form $v_0^{K^I} + V_{K^I} \subseteq \mathbb{F}_2^{|K^O|}$.

Since we want to use Theorem 4.9, we must consider the orthogonality of the vector subspaces Y, U_i and V_{K^I}. We will denote the dimensions of the relevant quotient spaces for the first Walsh transform as $t_i = \dim\left(Y/(Y \cap U_i^\perp)\right) = \dim\left(U_i/(U_i \cap Y^\perp)\right)$. For the last set of Walsh transforms, we will assume that these dimensions are constant for all the K^I, that is, $r_i = \dim\left(U_i/(U_i \cap V_{K^I}^\perp)\right) = \dim\left(V_{K^I}/(V_{K^I} \cap U_i^\perp)\right)$ for all $K^I \in \mathbb{F}_2^{|K^I|}$. Again, this assumption is not necessary, but it simplifies the complexity calculation.

The broad idea of the attack procedure is that, since the desired experimental correlation arrays $\widehat{\mathrm{cor}}(\cdot, K^I)$ are linear transformations of the distillation array A, we can compute them as the sum of l linear transformations of A. Each of these linear operations corresponds to the part of the linear map's spectrum which lies in the affine subspace $u_0^i + U_i \subseteq \mathbb{F}_2^{|K^O|}$. Each one of these transformations of A can be performed efficiently using the pruned Walsh transform algorithms. The full attack algorithm is the following:

1. **Distillation phase:** Since the first step of the pruned fast Walsh transform algorithm consists of combining the inputs into a smaller vector through addition, we might wish to merge it into the distillation phase in order to reduce the time and memory complexities. Depending on the relative sizes of N and $2^{\dim(Y)}$, we have two different options:

   - We can perform the distillation phase as usual (that is, we compute A in full), and then perform the first step of the pruned Walsh transform algorithm for each of the l pruned Walsh transforms separately to obtain l tables g_i of lengths 2^{t_i}. We note that we only need $2^{\dim(Y)}$ counters to store the full table. The cost of this operation is $\rho_D N + \rho_A \cdot l \cdot 2^{\dim(Y)}$.
   - We can instead construct l distilled tables g_i directly, without building the intermediate array A. The cost of this combined operation is $\rho_D \cdot l \cdot N$.

   In both cases, we require $\sum_{i=1}^{l} 2^{t_i}$ registers to store the resulting distilled data.

2. **Analysis phase:** We can once again save some operations and memory by mixing the last step of the first Walsh transform, the eigenvalue multiplication step, and the first step of the second set of Walsh transforms.

   (a) **First Walsh transform:** We first perform the (standard) fast Walsh transform algorithm on each of the arrays g_i which were constructed in the modified distillation phase, and obtain l arrays of the same sizes $\widehat{g_i}$. The time complexity of this operation is $\rho_A \sum_{i=1}^{d} t_i 2^{t_i}$.

   (b) **Walsh spectrum multiplication:** This step and the next will be repeated for every guess of K^I. We can consider each of the l subspaces $u_0^i + U_i$ separately. Inside each subspace, we go over all the non-zero Walsh coefficients. For each coefficient, we fetch the appropriate entry of $\widehat{g_i}$ (according to the last step of the pruned transform algorithm) and multiply it by the coefficient $\widehat{F}(u_0^i + u', K^I)$. The result is then added to the appropriate coordinate of an array h of length 2^{r_i}. This step has a cost of $(\rho_M + \rho_A)2^{|K^I|} \sum_{i=1}^{l} L_i$ and requires $\sum_{i=1}^{l} 2^{r_i}$ additional memory registers (assuming we can reuse the same memory from one K^I to the next). If we have stronger assumptions over the change in the Walsh coefficients of $F(\cdot, K^I)$ when K^I changes, for example when Corollary 5.8 applies, it may be possible to achieve further savings by performing the multiplication step a single time instead of once for every guess K^I.

   (c) **Second set of Walsh transforms:** We perform the fast Walsh transform on each of the h_i to obtain $\widehat{h_i}$, at a cost of $\rho_A \sum_{i=1}^{l} r_i 2^{r_i}$.

   (d) **Unfolding step:** Finally, for each possible guess of K^O, we can compute the experimental correlation of the linear approximation by adding l values (with appropriate signs), one from each of the $\widehat{h_i}$. This costs $\rho_A l 2^{|K^I|} 2^{\dim(V)}$.

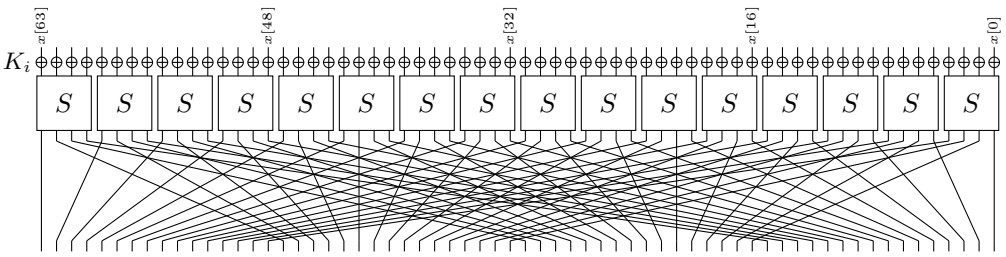


Figure 5.4: A round of PRESENT.

By adding up the cost of each step, we find that the total complexity of the algorithm which computes the experimental correlations is, after removing terms of lower order:

$$\underbrace{2^{|K^I|}}_{*} Nl + \underbrace{2^{|K^I|}}_{*} \sum_{i=1}^{l} t_i 2^{t_i} + \underbrace{2^{|K^I|}}_{**} \sum_{i=1}^{l} L_i + 2^{|K^I|} \sum_{i=1}^{l} r_i 2^{r_i} \text{ additions,} \tag{5.41}$$

$$\underbrace{2^{|K^I|}}_{**} \sum_{i=1}^{l} L_i \text{ products, and} \tag{5.42}$$

$$\sum_{i=1}^{l} 2^{t_i} + \sum_{i=1}^{l} 2^{r_i} \text{ memory registers,} \tag{5.43}$$

where the factors 2^{K^I} may be reduced to 1 depending on the attack. In particular, the factors indicated by $*$ can be removed when the non-zero Walsh coefficients of $F(\cdot, K^I)$ occupy the same subspaces $u_0^i + U_i$ independently from K^I, and the factors with $**$ can be removed when the Walsh coefficients of the different $\widehat{F}(\cdot, K^I)$ only differ by sign (see Corollary 5.8, for example).

5.4 Applications

After explaining our extensions and optimisations of the FFT linear key recovery attack of [CSQ07b] from a purely theoretical point of view, we will now proceed to illustrate them with three different examples. We will discuss the best known linear cryptanalysis of reduced-round PRESENT [BKL+07] and NOEKEON [DPVAR00], as well as the full-round DES [DES77].

5.4.1 Application to PRESENT

The first application example is reduced-round PRESENT [BKL+07]. We presented the first 28-round attacks on both 80-bit and 128-bit key variants in [FN20], and have also provided a 29-round attack on the 128-bit key version in [Fló22]. PRESENT is a good pedagogical example because it uses a permutation as its linear layer, which makes the analysis of the cost of the key recovery more straightforward.

5.4.1.1 PRESENT Specification and Previous Cryptanalysis

PRESENT [BKL+07] is a lightweight key-alternating block cipher which has received substantial attention from cryptanalysts since its introduction, and is a popular target for linear cryptanalysis, as it provides the best reduced-round attacks. It was made an ISO standard in 2012, and several symmetric primitives using similar constructions have been proposed, such as PUFFIN [CHW08], SPONGENT [BKL+11], RECTANGLE [ZBL+14], GIFT [BPP+17] and TRIFLE-BC [DGM+19].

PRESENT takes a 64-bit plaintext $x = x_{63} \dots x_0$ and an 80-bit (or 128-bit) key $K = \kappa_{79} \dots \kappa_0$ (or $K = \kappa_{127} \dots \kappa_0$) and returns a 64-bit ciphertext $y = y_{63} \dots y_0$. The encryption is performed by iteratively applying a round transformation to the state $b = b_{63} \dots b_0 = w_{15} | \cdots | w_0$, where each of the w_i represents a 4-bit nibble, $w_i = b_{4i+3} b_{4i+2} b_{4i+1} b_{4i}$. Both variants of PRESENT consist of 31 rounds, plus the addition of a whitening key at the output. Each round is the composition of three transformations:

- **addRoundKey:** Given the round key $K_i = \kappa_{63}^i \dots \kappa_0^i$, $1 \le i \le 32$ and the state b, the round key is xored bitwise to the state.
- **sBoxLayer:** A fixed 4-bit Sbox $S : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ is applied to each nibble w_i of the state. The Sbox S is given as a lookup table:

Algorithm 14: The PRESENT lightweight block cipher

Input: A plaintext x of 80 bits, a key K of 80 or 128 bits, a number of rounds r.
Output: A ciphertext y.
$y \leftarrow x$;
$\{K_i\}_{i=1}^{r+1} \leftarrow \text{KeySchedule}(K)$;
for $i \leftarrow 1$ **to** r **do**
  $y \leftarrow y \oplus K_i$;
  $y \leftarrow \text{sBoxLayer}(y)$;
  $y \leftarrow \text{pLayer}(y)$;
end
$y \leftarrow y \oplus K_{r+1}$;
return y;

Algorithm 15: Key schedule of PRESENT-80

Input: A master key K of 80 bits, a number of rounds r.
Output: $r+1$ round subkeys K_i of 64 bits.
$\kappa_{63}^{1} \dots \kappa_{0}^{1} \longleftarrow \kappa_{79} \dots \kappa_{16}$; // Extract first round subkey
for $i \leftarrow 1$ **to** r **do**
  $\kappa_{79} \dots \kappa_{0} \longleftarrow \kappa_{18} \dots \kappa_{19}$; // Rotate 19 bits to the right
  $\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76} \longleftarrow S(\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76})$; // S on leftmost nibble
  $\kappa_{19}\kappa_{18}\kappa_{17}\kappa_{16}\kappa_{15} \longleftarrow \kappa_{19}\kappa_{18}\kappa_{17}\kappa_{16}\kappa_{15} \oplus i$; // Add round counter
  $\kappa_{63}^{i+1} \dots \kappa_{0}^{i+1} \longleftarrow \kappa_{79} \dots \kappa_{16}$; // Extract round subkey
end
return $\{K_i\}_{i=1}^{r+1}$;

Algorithm 16: Key schedule of PRESENT-128

Input: A master key K of 128 bits, a number of rounds r.
Output: $r+1$ round subkeys K_i of 64 bits.
$\kappa_{63}^{1} \dots \kappa_{0}^{1} \longleftarrow \kappa_{127} \dots \kappa_{64}$; // Extract the first round subkey
for $i \leftarrow 1$ **to** r **do**
  $\kappa_{127} \dots \kappa_{0} \longleftarrow \kappa_{66} \dots \kappa_{67}$; // Rotate 61 bits to the left
  $\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124} \longleftarrow S(\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124})$;
  $\kappa_{123}\kappa_{122}\kappa_{121}\kappa_{120} \longleftarrow S(\kappa_{123}\kappa_{122}\kappa_{121}\kappa_{120})$; // S on 2 nibbles
  $\kappa_{66}\kappa_{65}\kappa_{64}\kappa_{63}\kappa_{62} \longleftarrow \kappa_{66}\kappa_{65}\kappa_{64}\kappa_{63}\kappa_{62} \oplus i$; // Add round counter
  $\kappa_{63}^{i+1} \dots \kappa_{0}^{i+1} \longleftarrow \kappa_{127} \dots \kappa_{64}$; // Extract i-th round subkey
end
return $\{K_i\}_{i=1}^{r+1}$;

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

- **pLayer:** A fixed bitwise permutation P is applied to the state b.

$$\begin{array}{rccc} P: & \{0,\dots,63\} & \longrightarrow & \{0,\dots,63\} \\ & j \neq 63 & \longmapsto & 16j \bmod 63 \\ & 63 & \longmapsto & 63 \end{array} \tag{5.44}$$

The key schedule is the only difference between the 80 and the 128-bit variants of PRESENT, which we will shorten to PRESENT-80 and PRESENT-128 for convenience. The full PRESENT algorithm for any number of rounds can be found in Algorithm 17. The key schedules are Algorithms 18 and 19.

PRESENT is a popular cipher in the community and has been the target of around 30 reduced-round cryptanalysis efforts, among which some of the most successful are linear attacks. Out of its 31 total rounds, Ohkuma found a weak-key linear attack on 24 [Ohk09]. Collard et al. found a statistical saturation attack on up to 26 rounds in 2009 [CS09]. Statistical saturation attacks were shown to be closely related to multidimensional linear approximations by Leander [Lea11]. Nakahara et al. proposed another 26-round linear attack [JSZW09], and Cho described a multidimensional attack with a larger success probability in 2010 [Cho10]. It wasn't until 2015 that 27 rounds were reached by Zheng et al. in [ZZ15]. A different 27-round attack was given by Bogdanov et al. [BTV18]. Our attacks are compared to the other linear attacks on 26 rounds or more in Table 5.1.

Table 5.1: Comparison of linear attacks on reduced-round PRESENT.

Key	Rds.	Complexity			P_S	Source
		Data	**Time**	**Memory**		
80	26	$2^{63.8}$ KP	$2^{72.0}$	$2^{32.0}$	51%	[Cho10, BN16]
		$2^{63.0}$ KP	$2^{68.6}$	$2^{48.0}$	95%	[BTV18]
		$2^{61.1}$ KP	$2^{68.2}$	$2^{44.0}$	95%	[FN20], 5.4.1.3
		$2^{60.8}$ KP	$2^{71.8}$	$2^{44.0}$	95%	[FN20], 5.4.1.3
	27	$2^{64.0}$ KP	$2^{74.0}$	$2^{67.0}$	95%	[ZZ15]
		$2^{63.8}$ DKP	$2^{77.3}$	$2^{48.0}$	95%	[BTV18]
		$2^{63.4}$ DKP	$2^{72.0}$	$2^{44.0}$	95%	[FN20], 5.4.1.3
	28	$2^{64.0}$ DKP	$2^{77.4}$	$2^{51.0}$	95%	[FN20], 5.4.1.4
128	28	$2^{64.0}$ DKP	2^{122}	$2^{84.6}$	95%	[FN20], 5.4.1.4
	29	$2^{64.0}$ DKP	$2^{124.06}$	$2^{99.2}$	67%	[Fló22], 5.4.1.5

Table 5.2: Linear Approximation Table (LAT) for the S-box of PRESENT. The entries with value 0 have been indicated with a − to facilitate reading. Furthermore, the entries corresponding to input and output masks of Hamming weight 1 have been highlighted.

Input mask α \ Output mask β	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
1	–	–	–	–	–	−4	–	−4	–	–	–	–	–	−4	–	4
2	–	–	2	2	−2	−2	–	–	2	−2	–	4	–	4	−2	2
3	–	–	2	2	2	−2	−4	–	−2	2	−4	–	–	–	−2	−2
4	–	–	−2	2	−2	−2	–	4	−2	−2	–	−4	–	–	−2	2
5	–	–	−2	2	−2	2	–	–	2	2	−4	–	4	–	2	2
6	–	–	–	−4	–	–	−4	–	–	−4	–	–	4	–	–	–
7	–	–	–	4	4	–	–	–	–	−4	–	–	–	–	4	–
8	–	–	2	−2	–	–	−2	2	−2	2	–	–	−2	2	4	4
9	–	4	−2	−2	–	–	2	−2	−2	−2	−4	–	−2	2	–	–
10	–	–	4	–	2	2	2	−2	–	–	–	−4	2	2	−2	2
11	–	−4	–	–	−2	−2	2	−2	−4	–	–	–	2	2	2	−2
12	–	–	–	–	−2	−2	−2	−2	4	–	–	−4	−2	2	2	−2
13	–	4	4	–	−2	−2	2	2	–	–	–	–	2	−2	2	−2
14	–	–	2	2	−4	4	−2	−2	−2	−2	–	–	−2	−2	–	–
15	–	4	−2	2	–	–	−2	−2	−2	2	4	–	2	2	–	–

5.4.1.2 Linear Approximations of PRESENT

Previous linear attacks on PRESENT such as [Ohk09, JSZW09, Cho10, ZZ15, BTV18] have all relied on the fact that the Sbox has eight linear approximations with correlation 2^{-3} and whose input and output masks have Hamming weight 1. These approximations, coupled with the fact that the linear layer is a bit permutation, lead to the existence of a multitude of linear trails with one active Sbox in each round. These trails conform linear hulls with high expected linear potential and input and output masks of Hamming weight 1. In our attacks, we will use three different sets of approximations with input and output masks of weights 1 and 2.

In order to find linear approximations and estimate their ELP, we will use the approach introduced by Albdelraheem in [Abd12], and which was explained in Subsection 2.2.2.3. We begin by computing the square correlation of all approximations of one round of PRESENT which: 1) only have up to two active Sboxes and 2) only activate up to two Sboxes in the previous and two Sboxes in the next round. There are 2800 input and 2800 output masks to the Sbox layer which verify these bounds on the number of active Sboxes, so a 2800×2800 truncated square correlation matrix was constructed. An approximation of the ELP of all the r-round linear approximations whose input and output masks are in this family can be obtained by multiplying this matrix by itself r times. This estimation accounts for all the linear trails which have at most two active Sboxes in every round.

The analysis of the resulting matrices showed that the linear approximations of PRESENT with the largest ELP only have a single active Sbox in the first and the last rounds. Table 5.3 contains

Table 5.3: An empirical classification of linear approximations of PRESENT with input and output masks of Hamming weight 1 or 2 according to their ELP. Indicated are the active Sbox of the first and last rounds, as well as the input and output masks of said Sboxes. Our three sets of approximations are indicated as I:*, II:$\circ$ and III:†.

Group	Input Mask	Input Sbox	Output Mask	Output Sbox	Qty.	ELP $r=22$	ELP $r=23$	ELP $r=24$
A	$A^{\dagger}_{\circ}$	$5^{\dagger}_{\circ}, 6^{\dagger}_{\circ}, 9^{\dagger}_{\circ}, 10^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$5^{\dagger}_{\circ}, 7^{\dagger}_{\circ}, 13^{\dagger}_{\circ}, 15^{\dagger}_{\circ}$	64	$2^{-59.9}$	$2^{-62.5}$	$2^{-65.1}$
B	$C^{\dagger}_{\circ}$	$5^{\dagger}_{\circ}, 6^{\dagger}_{\circ}, 9^{\dagger}_{\circ}, 10^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$5^{\dagger}_{\circ}, 7^{\dagger}_{\circ}, 13^{\dagger}_{\circ}, 15^{\dagger}_{\circ}$	64	$2^{-60.4}$	$2^{-63.0}$	$2^{-65.6}$
C1	$A^{\dagger}_{\circ}$	$5^{\dagger}_{\circ}, 6^{\dagger}_{\circ}, 9^{\dagger}_{\circ}, 10^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$6^{\dagger}_{\circ}, 9, 11, 14^{\dagger}_{\circ}$	64			
C2	A	$5, 6, 9, 10$	$4, 5$	$5, 7, 13, 15$	32	$2^{-60.6}$	$2^{-63.2}$	$2^{-65.8}$
C3	$A^{\dagger}_{\circ}$	$7^{\dagger}, 11^{\dagger}, 13^{\dagger}_{\circ}, 14^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$5^{\dagger}_{\circ}, 7^{\dagger}_{\circ}, 13^{\dagger}_{\circ}, 15^{\dagger}_{\circ}$	64			
D	$^{*}2^{\dagger}_{\circ}, ^{*}4^{\dagger}_{\circ}, 3, 5$	$^{*}5^{\dagger}_{\circ}, ^{*}6^{\dagger}_{\circ}, ^{*}9^{\dagger}_{\circ}, ^{*}10^{\dagger}_{\circ}$	$^{*}2^{\dagger}_{\circ}, ^{*}8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$^{*}5^{\dagger}_{\circ}, ^{*}7^{\dagger}_{\circ}, ^{*}13^{\dagger}_{\circ}, ^{*}15^{\dagger}_{\circ}$	256	$2^{-60.8}$	$2^{-63.4}$	$2^{-66.0}$
E1	$C^{\dagger}_{\circ}$	$5^{\dagger}_{\circ}, 6^{\dagger}_{\circ}, 9^{\dagger}_{\circ}, 10^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$6^{\dagger}_{\circ}, 9, 11, 14^{\dagger}_{\circ}$	64			
E2	C	5,6,9,10	4,5	5,7,13,15	32	$2^{-61.1}$	$2^{-63.7}$	$2^{-66.3}$
E3	$C^{\dagger}_{\circ}$	$7^{\dagger}, 11^{\dagger}, 13^{\dagger}_{\circ}, 14^{\dagger}_{\circ}$	$2^{\dagger}_{\circ}, 8^{\dagger}_{\circ}, 3^{\dagger}, 9^{\dagger}$	$5^{\dagger}_{\circ}, 7^{\dagger}_{\circ}, 13^{\dagger}_{\circ}, 15^{\dagger}_{\circ}$	64			
F1	A	$5, 6, 9, 10$	$2, 8, 3, 9$	10	16			
F2	A	$5, 6, 9, 10$	$4, 5$	$6, 9, 11, 14$	32			
F3	A	$5, 6, 9, 10$	$6, C$	$5, 7, 13, 15$	32	$2^{-61.3}$	$2^{-63.9}$	$2^{-66.5}$
F4	$A_{\circ}$	$7, 11, 13_{\circ}, 14_{\circ}$	$2_{\circ}, 8_{\circ}, 3, 9$	$6_{\circ}, 9, 11, 14_{\circ}$	64			
F5	A	$7, 11, 13, 14$	$4, 5$	$5, 7, 13, 15$	32			
F6	A	15	$2, 8, 3, 9$	$5, 7, 13, 15$	16			
G1	$^{*}2_{\circ}, ^{*}4_{\circ}, 3, 5$	$^{*}5_{\circ}, ^{*}6_{\circ}, ^{*}9_{\circ}, ^{*}10_{\circ}$	$^{*}2_{\circ}, ^{*}8_{\circ}, 3, 9$	$^{*}6_{\circ}, 9, 11, ^{*}14_{\circ}$	256			
G2	$2, 4, 3, 5$	$5, 6, 9, 10$	$4, 5$	$5, 7, 13, 15$	128	$2^{-61.5}$	$2^{-64.1}$	$2^{-66.7}$
G3	$8_{\circ}, 9$	$5_{\circ}, 6_{\circ}, 9_{\circ}, 10_{\circ}$	$2_{\circ}, 8_{\circ}, 3, 9$	$5_{\circ}, 7_{\circ}, 13_{\circ}, 15_{\circ}$	64			
G4	$^{*}2_{\circ}, ^{*}4_{\circ}, 3, 5$	$7, 11, ^{*}13_{\circ}, ^{*}14_{\circ}$	$^{*}2_{\circ}, ^{*}8_{\circ}, 3, 9$	$^{*}5_{\circ}, ^{*}7_{\circ}, ^{*}13_{\circ}, ^{*}15_{\circ}$	256			

Table 5.4: The capacities of our three sets of approximations.

	# Approx.	Capacity ($r=22$)	Capacity ($r=23$)	Capacity ($r=24$)
I (*)	128	$2^{-54.11}$	$2^{-56.71}$	$2^{-59.31}$
II ($\circ$)	296	$2^{-52.60}$	$2^{-55.20}$	$2^{-57.80}$
III (†)	448	$2^{-51.78}$	$2^{-54.38}$	$2^{-56.98}$

a classification of approximations with one active Sbox in the first and the last round and masks of Hamming weight 1 or 2, according to their ELP. From these approximations, we have selected three different sets as linear distinguishers, considering both their total linear capacity and the number of keybits involved in a two-round key-recovery.

Set I, with 128 approximations, has the lowest capacity, but only uses masks of Hamming weight 1 and has a cheaper key recovery than the others. Set III has 448 approximations and the largest capacity but requires guessing a lot of bits in the key recovery, as it has approximations with both masks of Hamming weight 2. Set II is an intermediate where masks of Hamming weight 2 are only used in the input. The capacity for these three sets can be found in table 5.4. We have also estimated the advantage that is obtained by these sets of approximations using the statistical model which was explained in Section 2.4.

These approximations are not statistically independent (as they are not even linearly independent). One possible solution would be the application of multidimensional linear cryptanalysis. However, this would consider all the linear combinations of the approximations, and the benefits of the masks of low Hamming weight would be lost. Instead, we use the multiple linear cryptanalysis statistic, and we have estimated the probability of success under the assumption that the approximations are statistically independent using the methods of Blondeau et al. [BN17]. In order to justify the validity of the resulting estimations, we provide experimental results which conform to the theoretical predictions for a reduced number of rounds. Figures 5.5 and 5.6 contain the advantage predictions for the 22, 23 and 24 round distinguishers as well as experiments for 6 and 8 round distinguishers.

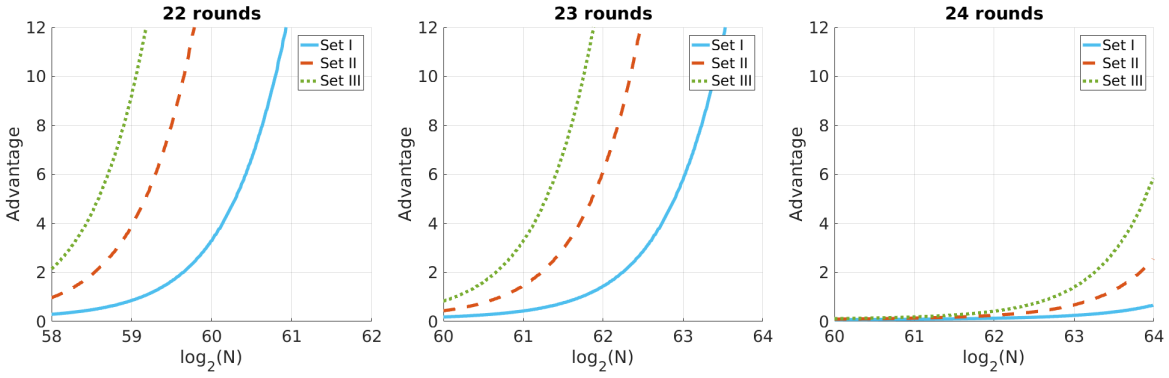


Figure 5.5: Advantage obtained by each of our sets of approximations for 22, 23 and 24 rounds of PRESENT with 0.95 probability in a distinct known plaintext scenario.

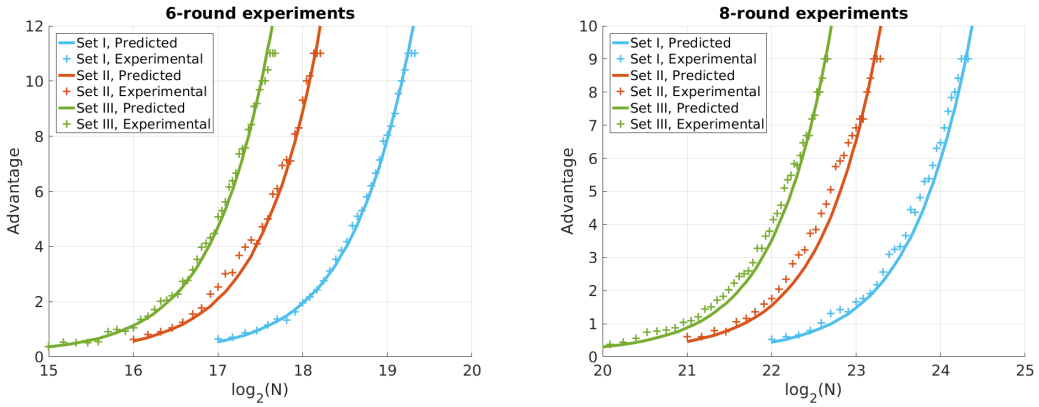


Figure 5.6: Experimental advantage for attacks on 10 and 12 rounds of PRESENT using linear distinguishers over 6 and 8 rounds, respectively.

The experimental verification was performed by simulating attacks on 10 and 12 rounds with key recovery in the first and last two rounds. The multiple linear cryptanalysis statistic of the right key was compared against a random sample of 2^{12} (10-round attack) or 2^{10} (12-round attack) keys. The position of the right-key statistic among these provides an estimation of the advantage of up to 12 or 10 bits. This was repeated for 20 different random right keys and 20 different random data samples for each value of N, providing a sample of 400 values of the advantage. The 5th percentile was used as an estimation of the advantage that's achieved with probability 0.95.

5.4.1.3 Attacks on 26 and 27-round PRESENT-80

The first attacks on PRESENT that we propose are based on set I of linear approximations, and use the key recovery algorithm of Subsection 5.2.2, without any pruning of the Fast Walsh transforms. Since this set is only effective on up to 23 central rounds and the attack will perform a key recovery on the first two and last two rounds, the attack is effective on up to 27 rounds. In order to describe these attacks more easily, we make use of the following properties of the bit permutation:

Proposition 5.9 (Key recovery on PRESENT). *Let $\hat{x}$ be the state at the beginning of the second round of PRESENT. Given two fixed values of i, j between 0 and 3, the four bits*

$$\hat{x}_{48+4i+j},\ \hat{x}_{32+4i+j},\ \hat{x}_{16+4i+j},\ \hat{x}_{4i+j}$$

can be obtained from

- *the 16 bits of the plaintext $x_{16j+15} \dots x_{16j}$,*
- *the 16 bits of the first round subkey $\kappa^1_{16j+15} \dots \kappa^1_{16j}$,*
- *and the 4 bits of the second round subkey $\kappa^2_{16i+4j+3}\kappa^2_{16i+4j+2}\kappa^2_{16i+4j+1}\kappa^2_{16i+4j}$.*

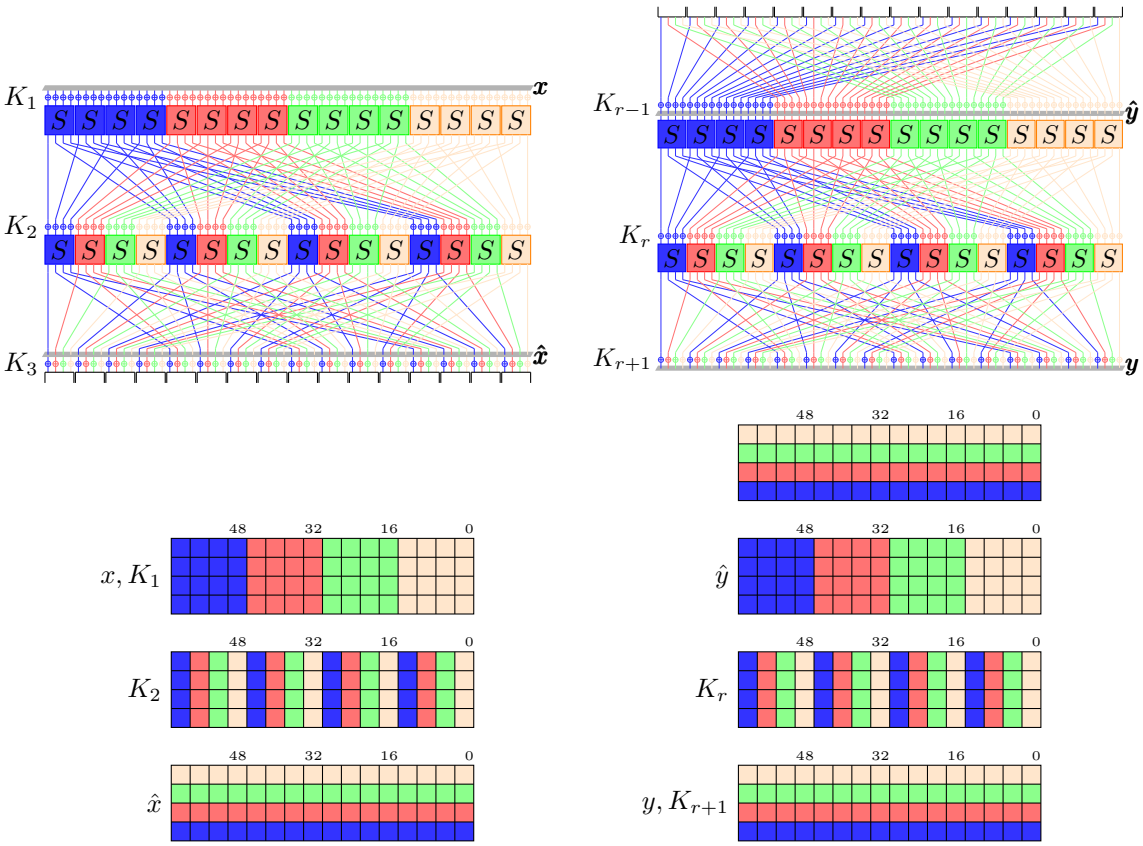


Figure 5.7: The four groups of bits for the key-recovery on the first and last rounds.

Let $\hat{y}$ be the state before the application of sBoxLayer in the $(r-1)$-th round of PRESENT. Given two fixed values of i, j between 0 and 3, the four bits

$$\hat{y}_{P(16j+12+i)},\ \hat{y}_{P(16j+8+i)},\ \hat{y}_{P(16j+4+i)},\ \hat{y}_{P(16j+i)}$$

can be obtained from

- *the 16 bits of the ciphertext $y_{60+i}, y_{56+i}, \ldots, y_{4+i}, y_i$,*
- *the 16 bits of the last round subkey $\kappa^{r+1}_{60+i}, \kappa^{r+1}_{56+i}, \ldots, \kappa^{r+1}_{4+i}, \kappa^{r+1}_{i}$,*
- *and the 4 bits of the previous round subkey $\kappa^{r}_{48+4i+j}\kappa^{r}_{32+4i+j}\kappa^{r}_{16+4i+j}\kappa^{r}_{4i+j}$.*

With the help of this result, we can mount key-recovery attacks on up to 27-round PRESENT-80 by extending approximation set I with two rounds of key-recovery at both the top and bottom of the cipher using our multiple linear cryptanalysis key-recovery algorithm from Subsection 5.2.2. We can use a two part description of the key recovery map (one part from the input side and one part from the output side), with $f^i_0 = 0$ for all approximations. The details on the key schedule for 26 and 27 rounds can be found in figures 5.8 and 5.9. In particular

$$\begin{aligned} &M_1 = 4,\ M_2 = 16,\ |K^O| = 32 + 32 = 64,\ |K^I| = 12 + 12 = 24, \\ &|K^{O,i}| = 16 + 16 = 32,\ |K^{I,i}| = 4 + 4 = 8 \text{ for all 128 approximations}, \\ &|k_T| = 61 \text{ for 26 rounds},\ |k_T| = 68 \text{ for 27 rounds}. \end{aligned} \tag{5.45}$$

A simple lower bound on the cost of an r-round PRESENT encryption ρ_E is $2 \cdot 64 \cdot r + 64$ binary operations (since each round requires at the very least adding the round subkey and writing each output bit for sBoxLayer). For 26 rounds, this is 3392 binary operations. On the other hand, $\rho_A \simeq 128$, $\rho_M \simeq 3 \cdot 64^{\log_2(3)} \simeq 2143$. Plugging these values into Formula 5.33, we find that the time complexity of the analysis phase should be at most 2^{65} full encryptions for 26 rounds and 2^{72} full encryptions for 27 rounds. The search phase time complexity depends on the available data and can be estimated thanks to the graphs in Figure 5.5. The complexities of both attacks are given in Table 5.1. These attacks can be easily extended to PRESENT-128.

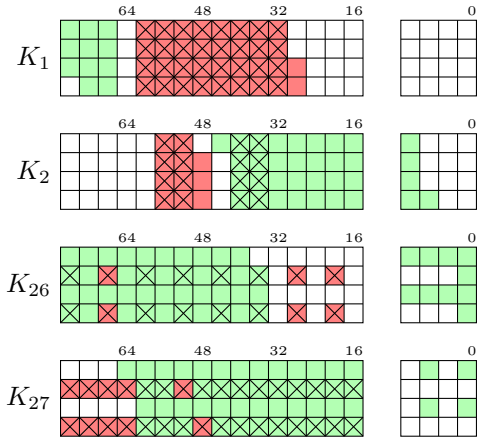


Figure 5.8: Using the key schedule in the key recovery attack on 26-round PRESENT-80 using approximation set I. In total there are 96 bits of the subkeys which need to be guessed, which have been indicated by a cross. However, they can all be deduced from the $|k_T| = 61$ bits of key which have been highlighted in (dark) red. From these bits of key, all the bits in (light) green can be extracted, which includes all the necessary bits for the attack.

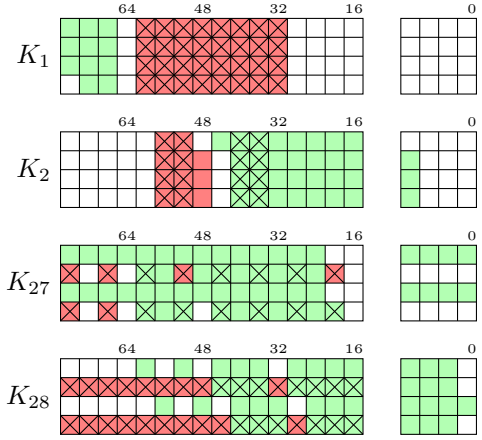


Figure 5.9: Using the key schedule in the key recovery attack on 27-round PRESENT-80 using approximation set I, with $|k_T| = 68$.

5.4.1.4 Attacks on 28-round PRESENT-80 and PRESENT-128

Approximation sets II and III can also be extended by two rounds of key recovery at both sides to construct attacks on up to 28-round PRESENT. As set III has a larger capacity but requires an expensive key recovery with many active keybits, we found that set II is best suited to attack PRESENT-80 and set III gives better results on PRESENT-128, where there is more time complexity margin for key guessing.

The parameters for an attack using approximation set II on PRESENT-80, with the key-schedule analysis represented in figure 5.10 are:

$$\begin{aligned} &M_1 = 8,\ M_2 = 32,\ |K^O| = 48 + 32 = 80,\ |K^I| = 24 + 16 = 40, \\ &|K^{O,i}| \leq 32 + 16 = 48,\ |K^{I,i}| \leq 8 + 4 = 12,\ \text{for all 296 approximations},\ |k_T| = 73. \end{aligned} \tag{5.46}$$

There are 160 approximations for which the input and output masks have weight 1. For each of these approximations, we have $|K^{O,i}| = 16 + 16 = 32$ and $|K^{I,i}| = 4 + 4 = 8$. Computing the associated 2^{40} experimental correlations has cost $32 \cdot 2^{8+32} = 2^{45}$ operations for each approximation, which is negligible in comparison to the rest of the attack.

For the remaining approximations, the cost should be $48 \cdot 2^{12+48} = 2^{65.58}$ operations, which will be a little too costly. For this reason, we will perform some simple pruning in the last set of Walsh transforms, using some keybit values which are determined by the key schedule. We note that all these approximations have an input Sbox mask A or C. A look at the key-recovery diagrams shows that at

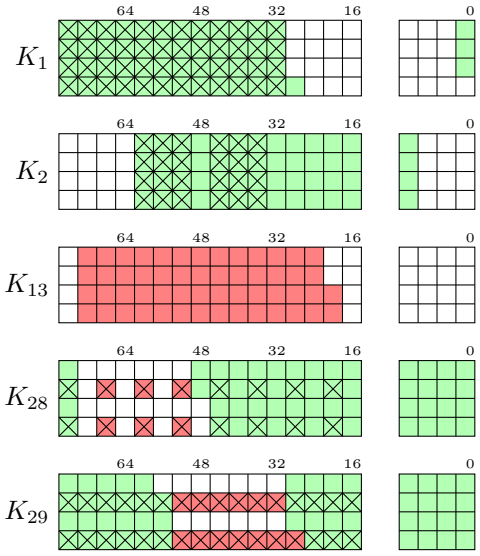


Figure 5.10: Key-recovery on 28-round PRESENT-80 using approximation set II, with $|k_T| = 73$.

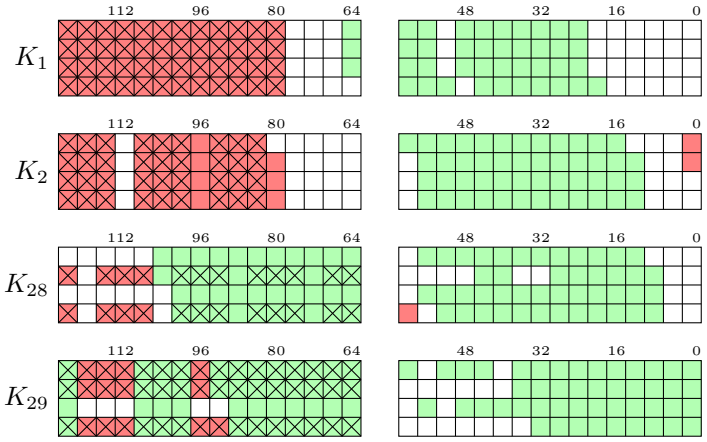


Figure 5.11: Key-recovery on 28-round PRESENT-128 using approximation set III ($|K_T| = 114$).

least 5 active bits of K_1 can be deduced from the active bits of K_2. Furthermore, some bits of K_{29} can be deduced from K_1 and K_2. If we consider the matrix description of the two part attack, we can prune the Walsh Transforms corresponding to the matrices B^{k_1} (input side), thus reducing their cost from 2^{37} to $2^{32.9}$ operations. It also means that the memory requirement for each approximation is reduced by a factor of 2^5. After this first pruning, the transforms associated with the last two rounds (or the matrices C^{k_2}) can also be pruned, reducing the complexity of each transform to 2^{16}. This allows to keep the time complexity of the analysis phase below 2^{77} full PRESENT encryptions, and reduces the memory cost to 2^{51} registers.

For an attack using approximation set III on PRESENT-128, with the key-recovery part represented in figure 5.11 we have:

$$\begin{aligned} &M_1 = 16,\ M_2 = 96,\ |K^O| = 48 + 48 = 96,\ |K^I| = 36 + 36 = 72, \\ &|K^{O,i}| \leq 32 + 32 = 64,\ |K^{I,i}| \leq 8 + 8 = 16,\ \text{for all 448 approximations},\ |k_T| = 114. \end{aligned} \tag{5.47}$$

This means that the time complexity of the analysis phase of the attack should be smaller than 2^{121}. The memory is mainly devoted to the condensed correlation tables corresponding to the largest value of $|K^{O,i}| + |K^{I,i}|$, that is, for approximations which require 80 bits of subkey to be guessed. Since the correlation of these approximations can be condensed into 18 tables, we conclude that the memory cost is $18 \cdot 2^{80} \simeq 2^{84.6}$ memory registers of 64 bits. The complexities of both 28-round attacks can be found in Table 5.1.

We note that in the 28-round attacks in the table we have considered that the full codebook is available, but it is possible to consider different trade-offs between the available data and the time complexity of the

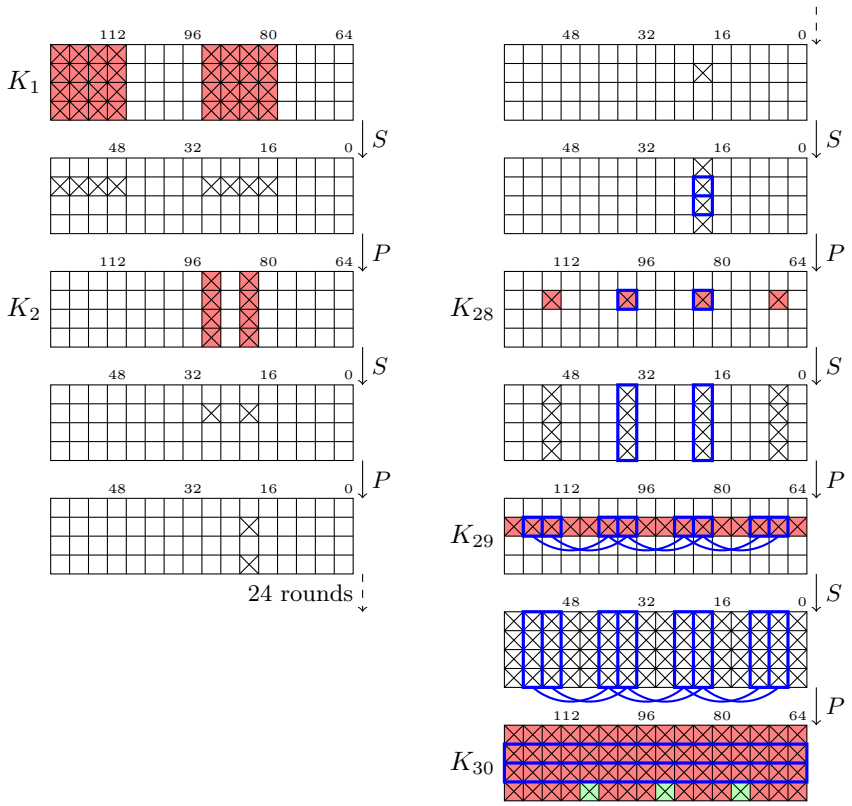


Figure 5.12: Key recovery for one approximation in the 29-round PRESENT-128 attack.

exhaustive search. For instance, in the case of PRESENT-128, if $N = 2^{63.5}$ distinct plaintext-ciphertext pairs are available, the advantage is 2.8 bits. This translates into an attack with $2^{125.2}$ time complexity.

5.4.1.5 Attack on 29-round PRESENT-128

We will now describe an attack on 29-round PRESENT-128 which is part of [Fló22]. It uses approximation set II over 24 rounds between the 3rd and the 26th rounds. This means we will perform two rounds of key recovery in the plaintext side and three rounds of key recovery in the ciphertext side. It makes use of the full pruned FWT key recovery algorithm which was described in Section 5.3.

We consider the computation of the experimental correlations for an example 24-round linear approximation with input mask `0000000000A00000` and output mask `0000000000200000`. We will now compute the time complexity of obtaining the experimental correlation of this approximation for all guesses of the active keybits using the pruned Walsh transform algorithm. For comparison purposes, we first compute the cost using the Walsh transform without any kind of pruning. There are 32 active keybits in K_1, 8 active keybits in K_2, 4 active keybits in K_{28}, 16 active keybits in K_{29}, and 64 active keybits in K_{30} (see the crossed-out bits in Figure 5.12). These add up to a total inner key guess of 28 bits and an outer key guess of 96 bits. We will thus require 2^{96} memory registers, and the time complexity will be around $96 \cdot 2^{96+28} \simeq 2^{130.6}$ additions/subtractions, which gives very little margin to iterate this procedure over multiple approximations with a total time complexity below 2^{128} equivalent encryptions.

In order to reduce this cost as much as possible, we will consider both the structure of the key recovery map and the keybit dependency relationships in order to prune both stages of Walsh transforms in the FFT key recovery algorithm. The key recovery map consists of three parts which are fully independent from each other (if we ignore the key schedule), each one of which corresponds to one of the three active bits in the input and output masks.

Both parts corresponding to the input mask are essentially identical. If we denote by S_1 the second component of S (that is, the second output bit) and the four input nibbles by x_0, x_1, x_2, x_3, then these $\mathbb{F}_2^{16} \times \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2$ maps are of the form

$$F(x_3, x_2, x_1, x_0; k) = S_1((S_1(x_3), S_1(x_2), S_1(x_1), S_1(x_0)) \oplus k^I). \tag{5.48}$$

According to Corollary 5.6, each Walsh coefficient of these maps is the product of up to five coefficients

Table 5.5: Restricted Walsh spectra used in the key recovery on 29-round PRESENT-128.

	v	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(2,\cdot)$	$\mathcal{X}=\varnothing$	0	0	4	4	−4	−4	0	0	4	−4	0	8	0	8	−4	4
	$\mathcal{X}=\{3,5,B,D\}$	0	0	4	4	−4	−4	0	0	0	0	0	8	0	8	0	0
	v	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(4,\cdot)$	$\mathcal{X}=\varnothing$	0	0	−4	4	−4	−4	0	8	−4	−4	0	−8	0	0	−4	4
	$\mathcal{X}=\{1,3,D,F\}$	0	0	0	0	−4	−4	0	8	−4	−4	0	−8	0	0	0	0
	v	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(8,\cdot)$	$\mathcal{X}=\varnothing$	0	0	4	−4	0	0	−4	4	−4	4	0	0	−4	4	8	8
	$\mathcal{X}=\{0,1,2,4,5,7,9,C\}$	0	0	0	−4	0	0	−4	0	0	4	0	0	−4	0	8	0

of the Walsh spectrum of S_1, $\widehat{S}(\cdot,2)$, which has six zeros.

$$\widehat{F}(u_3,u_2,u_1,u_0;k)=\widehat{S}(w(u_3,u_2,u_1,u_0),2)\cdot\widehat{S}(u_3,2)\cdot\widehat{S}(u_2,2)\cdot\widehat{S}(u_1,2)\cdot\widehat{S}(u_0,2). \tag{5.49}$$

We now look at the part corresponding to the output mask, which is a map $\mathbb{F}_2^{64}\times\mathbb{F}_2^{20}\longrightarrow\mathbb{F}_2$ with a similar structure to the previous ones but over three rounds. In round 27, the Walsh coefficient $\widehat{S^{-1}}(\mathtt{F},2)=\widehat{S}(2,\mathtt{F})=4$ has a large impact on the number of non-zero Walsh coefficients of the map. By removing the plaintexts which lead to an input 3, 5, B or D to this inverse Sbox, this coefficient becomes zero. We can thus split all non-zero Walsh coefficients of this part of the map into two affine subspaces of dimension 48 (instead of a single space of dimension 64). Each affine subspace corresponds to one non-zero coefficient of Hamming weight 3: $\widehat{S^{-1}}_{x\in\mathbb{F}_2^4\setminus\mathcal{X}}(\mathtt{B},2)$ and $\widehat{S^{-1}}_{x\in\mathbb{F}_2^4\setminus\mathcal{X}}(\mathtt{D},2)$. The "inactive" bits in each of these subspaces have been surrounded by a thicker outline in Figure 5.12. The cost of this modification is a reduction of the available data by a factor of $3/4=2^{-0.42}$.

The other aspect of the key recovery which can be used to reduce the time complexity is the key schedule. For the purposes of pruning the Walsh transforms, we prefer keybit relationships which are linear or which can describe outer active keybits in terms of inner active keybits. We thus begin by guessing the 28 inner keybits. There are three bits of K_{30} (part of the outer key guess) which can be deduced from K_{28} (inner key guess).

We can now compute the time complexity of evaluating the experimental correlation of this approximation for each key guess. The first Walsh transform of the analysis phase is separated into two, and each one is pruned at the output to an affine subspace of dimension $48+32=80$. The distillation phase, which is combined with the first step of the pruned transforms, costs $2N$ memory accesses and requires $2\cdot 2^{80}$ memory registers. The time complexity of the reduced dimension Walsh transforms is $2\cdot 80\cdot 2^{80}\simeq 2^{87.32}$ additions. The cost of multiplying by the non-zero Walsh coefficients can be estimated to be less than $\left(\frac{10}{16}\right)^{20}\cdot 2\cdot 2^{80}\simeq 2^{67.44}$ multiplications, where the origin of the 10/16 factor is the fact that $\widehat{S}(2,\cdot)$ and $\widehat{S^{-1}}(\cdot,2)$ have six zeroes.

The second pair of Walsh transforms has to be repeated once for each of the 2^{28} inner key guesses. For each one, we must perform two Walsh transforms which are pruned on the input side to subspaces of dimension 80 and on the output side to subspaces of dimension 93. It can be shown that the dimension of $X/(X\cap U^{\perp})$ is minimal and equal to 77. Thus, the cost of this second pair of fast Walsh transforms for all inner key guesses is thus $2^{28}\cdot 2\cdot 77\cdot 2^{77}\simeq 2^{112.27}$ additions. The cost of combining both arrays into the correlation for each key guess is 2^{120} additions. However, this complexity can be reduced by a great amount by considering the key schedule. Indeed, 25 bits of the guess at K_1 and three bits of K_{28} can be deduced from the guess at K_{30}, thus reducing the number of necessary additions to just 2^{92}.

We would like to draw the reader's attention towards the fact that we have achieved a decrease in the time complexity by a factor of almost 2^{16} by increasing the data complexity by a much smaller factor of $2^{0.42}$, illustrating that careful filtering of the data can lead to very significant gains.

We will now provide an overview of the full attack using the 296 approximations of set II. We will consider two types of linear approximations depending on the Hamming weight of the input mask, as their contribution to the time complexity will be different:

- **Type I (groups D,G):** These are the 160 approximations which have input masks of Hamming weight 1. The key recovery for these approximations involves 16 bits of K_1, 4 bits of K_2, 4 bits of

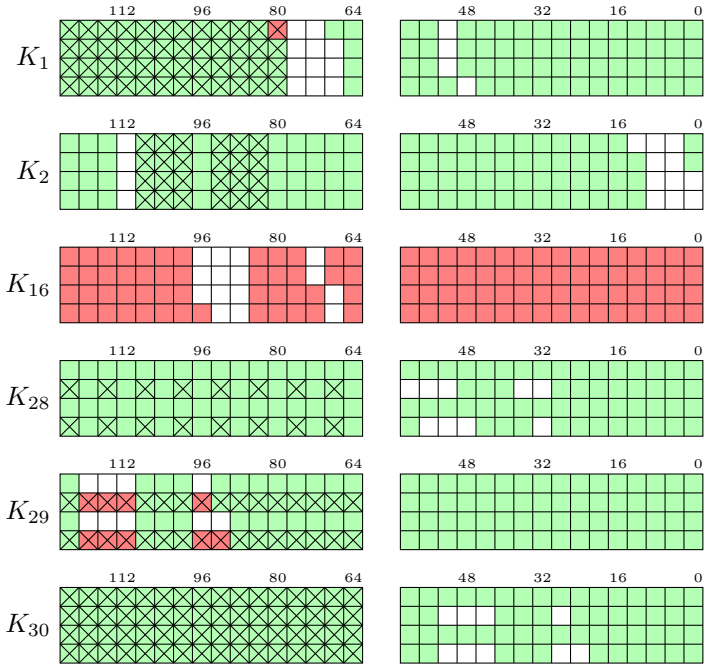


Figure 5.13: Determining all the active keybits for all approximations (crossed out in the figure) with as few guesses as possible. We can deduce the (light) green bits if we know the 123 keybits highlighted in (dark) red.

K_{28}, 16 bits of K_{29} and 64 bits of K_{30}. This means we can compute the experimental correlations for each of these approximations with around $2^{24} \cdot 80 \cdot 2^{80} \simeq 2^{110.32}$ additions and subtractions without any pruning.

- **Type II (groups A,B,C,E,F):** These are the 136 approximations with input masks of Hamming weight 2. There are 16 active bits in K_1, 8 bits in K_2, 4 bits in K_{28}, 16 bits in K_{29}, and 64 bits in K_{30}. We treat these approximations as in the example approximation: we study the Walsh spectrum of the active Sbox in round 27 (see Table 5.5).
  - For 48 approximations, the desired input bit to this Sbox corresponds to the mask 2, and we can split the Walsh spectrum of the key recovery map into two affine subspaces of dimension 80 by discarding 1/4 of the data.
  - For 40 approximations, the desired input bit is 4, and we can split the spectrum into 2 spaces of dimension 80 by discarding 1/4 of the data.
  - For the other 48 approximations, the desired input bit to this Sbox corresponds to the mask 8, in which case the spectrum lies on a subspace of dimension 80 after discarding 1/2 of the data.

  For all these approximations, we can compute the experimental correlation with either $2 \cdot 2^{28} \cdot 80 \cdot 2^{80} \simeq 2^{115.32}$ or $2^{28} \cdot 80 \cdot 2^{80} \simeq 2^{114.32}$ additions/subtractions. We ignore possible further reductions in time complexity given by pruning at the outputs using the key schedule, as they may be different for each approximation. In total, $2^{123.08}$ additions are required. For each approximation, we also have to combine two arrays into the experimental correlations at a cost of 2^{120} additions. However, using the key schedule, we can reduce this number of additions to at most 2^{92} per approximation.

We next compute the parameters of the algorithm related to computing the multiple linear cryptanalysis statistic. We have $M_1 = 4$, where two groups are formed by approximations of type I and two groups are formed by approximations of type II. We have $|M_2| = 32$, with 8 groups of approximations of type I and 16 groups of approximations of type II. Considering the key schedule, for the Type II approximations we need to guess at most 92 bits, and for the Type I at most 88 bits. We can thus perform the first combination step with $136 \cdot 2^{92} + 160 \cdot 2^{88} \simeq 2^{99.2}$ products and the same number of additions.

Finally, we combine these squared experimental correlations into a single value of the multiple linear cryptanalysis statistic for each guess of the master key, k_T. Figure 5.13 illustrates an efficient guess of 123 keybits from which all the active keybits in the attack can be deduced. The cost of this step is

$32 \cdot 2^{123} = 2^{128}$ additions. After this we can perform an exhaustive search over the five remaining keybits which will have a time complexity of 2^{123} encryptions if we aim for a five-bit advantage.

We now look at the complexity of the attack:

- **Data complexity:** We used the model by Blondeau and Nyberg [BN17] which was discussed in Section 2.4. The model must be modified slightly in order to accommodate the fact that the number of available plaintexts now depends on the approximation. We find that if the whole codebook is available (2^{64} distinct known plaintexts), we obtain 5 bits of advantage with a 67% probability.
- **Time complexity:** In this case, the dominant part of the time complexity is the computation of the final multiple linear cryptanalysis statistic and the exhaustive key search. The latter requires 2^{123} PRESENT encryptions because of the 5-bit advantage. The former requires 2^{128} additions. If we assume that a sum requires at most 128 bit operations and a 29-round PRESENT encryption requires at least 3776, the additions will be equivalent to at most $2^{123.18}$ 29-round PRESENT encryptions. The total time complexity of the attack is thus $2^{124.06}$ encryptions.
- **Memory complexity:** In the distillation phase, $160 \cdot 2^{16+64} + 136 \cdot 2 \cdot 2^{32+48} \simeq 2^{88.75}$ registers are required. The 32 intermediate multiple linear cryptanalysis statistic tables amount to $2^{99.2}$ memory registers, which dominate the memory complexity of the attack.

5.4.2 Application to the DES

We now present a variant of Matsui's linear attack [Mat93, Mat94a] on the Data Encryption Standard [DES77] which was presented in [Fló22]. The idea of our improved attack is to remove the last round from Matsui's linear approximation and to cover it with an additional key recovery round. We will use a 13-round linear approximation which is identical to the 14-round linear approximation which is used in [Mat94a] except for the last round. This increases the correlation from $-2^{-19.75}$ to $2^{-19.07}$. The input mask is $(\texttt{00000000}, \texttt{01040080})$ at (L_1, R_1) and the output mask is $(\texttt{21040080}, \texttt{00000000})$ at (L_{14}, R_{14}). This variant has lower data complexity than the original ($2^{41.5}$ vs. 2^{43}), but has a larger memory complexity ($2^{38.75}$ vs. $2^{26.00}$) due to the larger key guess. It also compares favourably in terms of data complexity against the conditional linear cryptanalysis variant of the attack introduced by Biham and Perle [BP18], which has 2^{42} data complexity.

Figure 5.14 indicates the active keybits in the key recovery in rounds 1, 15, and 16. There are 40 active keybits in total: 3 are active in round 1, one is active in round 15, 28 are active in round 16, 3 are active in both rounds 1 and 16, and 5 are active in both rounds 15 and 16. All active keybits are represented as part of K, after applying the appropriate key schedule bit rotation. There are 44 active plaintext/ciphertext bits (four of which are duplicated before the key addition because of the expansion map). All the active keybits are xored directly with plaintext or ciphertext material. In short, we have $|K^O| = 48$ and $|K^I| = 0$. An attack using the same version of Algorithm 2 as [Mat94a] would have a time complexity of $O(N) + 2^{44+40} \simeq 2^{84}$ operation. An attack based on the FFT without any kind of internal optimisation or pruning would require $O(N) + 48 \cdot 2^{48}$ operations.

5.4.2.1 The Walsh Spectrum of the Key Recovery Map

Figure 5.15 shows the full key recovery map for the attack, including all the key material. In other words, it shows how the value of the linear approximation is computed from the plaintext, ciphertext, and key. Our aim is to identify the zeroes in this function's Walsh spectrum. We note that all key material is xored to the plaintext/ciphertext, and that there are seven plaintext/ciphertext bits which are xored at the end and can be considered separately as the term f_0. The rest of the map consists of two independent parts if we ignore the key schedule: one corresponds to the first round and the other corresponds to the last two rounds.

In the case of the map for the first round, which we will denote by f_1, we can see that it consists of the application of S_5 and the xoring of three of its output bits. If we look at the Walsh spectrum of S_5, we can see that for the output $y_1 \oplus y_2 \oplus y_3$ we have 50 non-zero coefficients out of the total 64.

The map for the last two rounds f_2 is a little more complex, as it covers two rounds instead of one. It is the composition of three maps: the first is an $\mathbb{F}_2^{42} \to \mathbb{F}_2^{12}$ map consisting of the application of the six active Sboxes in round 16 (selecting a single output bit for each), as well as the identity on the six active bits on the left part of the ciphertext. We then apply a linear $\mathbb{F}_2^{12} \to \mathbb{F}_2^6$ map which xors the outputs of the six sboxes into the untouched ciphertext material. Finally, we apply S_5 and xor the four outputs. If we look at the Walsh spectrum of S_5 with output mask $\texttt{F}$, we note that there are 32 zeros, one of them corresponding to the input mask $\texttt{3F}$. Table 5.7 shows this specific column of the Walsh spectrum.

Table 5.6: Comparison of selected attacks on the Data Encryption Standard.

	Complexity				
Type	**Data**	**Time**	**Memory**	P_S	**Source**
Differential Cryptanalysis	$2^{47.00}$ CP	$2^{37.00}$	$\mathcal{O}(1)$	58%	[BS92]
Linear Cryptanalysis	$2^{43.00}$ KP	$2^{39.00}$	$2^{26.00}$	50%	[Mat94a]
Multiple linear Cryptanalysis	$2^{42.78}$ KP	$2^{38.86}$	$2^{30.00}$	85%	[BV17]
Conditional Linear Cryptanalysis	$2^{42.00}$ KP	$2^{42.00}$	$2^{28.00}$	90%	[BP18]
Linear Cryptanalysis	$2^{41.50}$ KP	$2^{42.13}$	$2^{38.75}$	70%	[Fló22]

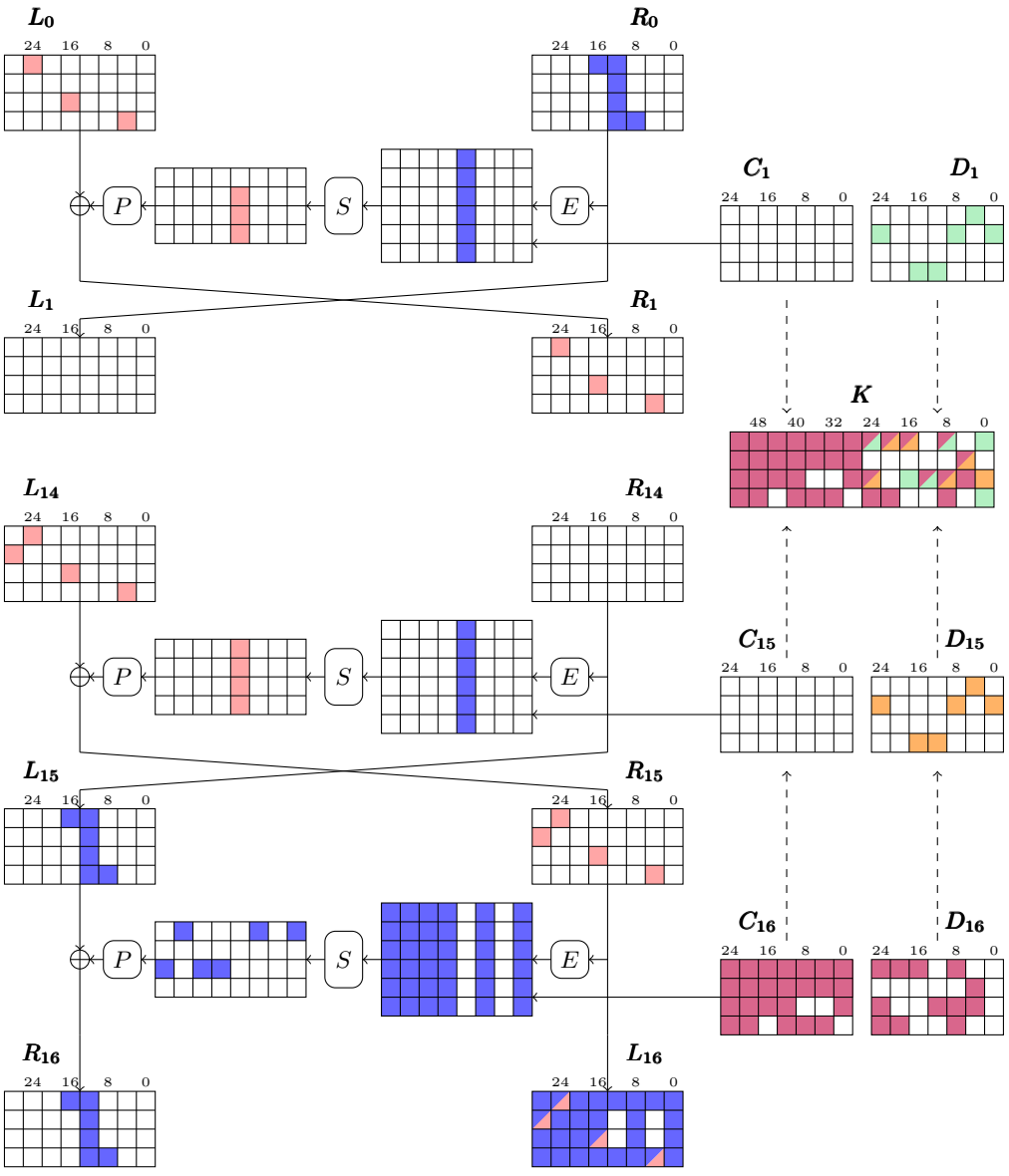


Figure 5.14: Key recovery in rounds 1, 15 and 16 of the DES. States are represented as divided into nibbles, except for those before the Sbox layer which are divided in groups of 6 bits. The least significant bit is the one on the upper right. ■ represents a bit which appears linearly in the linear approximation, while ■ represents any other active (nonlinear) bit. ■, ■, and ■ represent keybits which are active in rounds 1, 15 and 16, respectively.

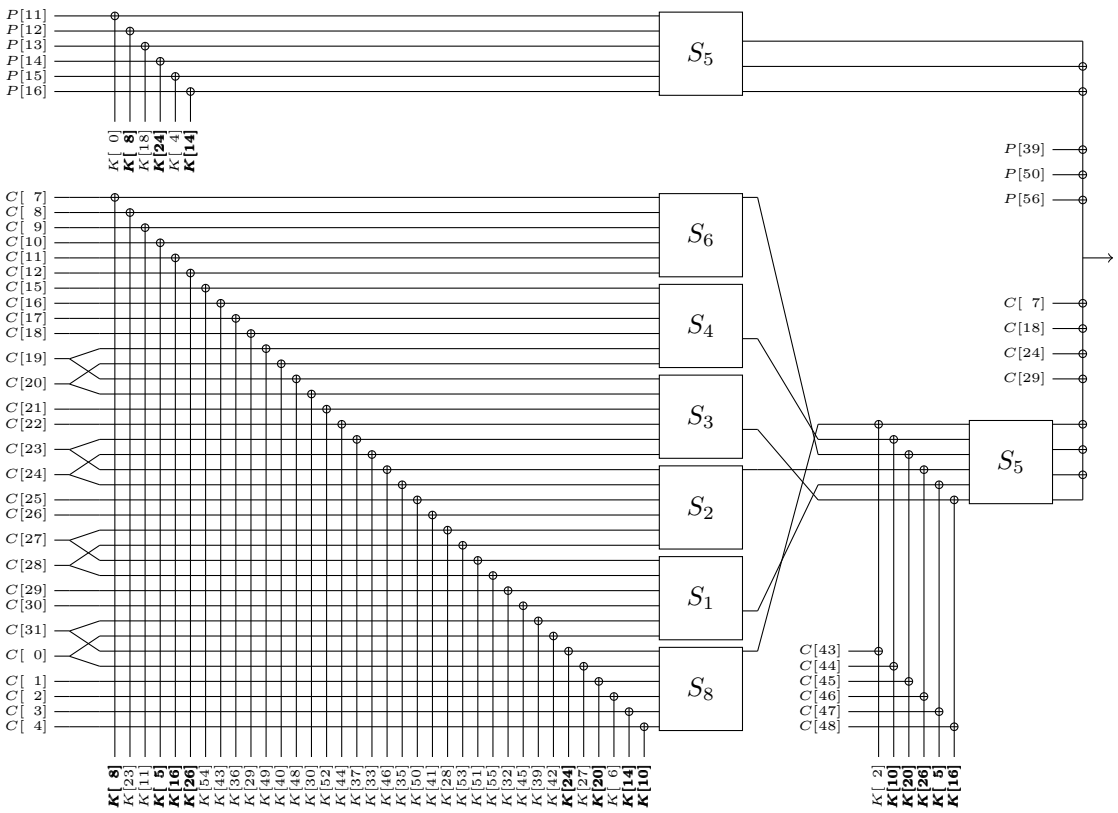


Figure 5.15: Schematic of the key recovery map for the DES attack.

We consider the coefficients $\widehat{f_2}(u_0, \ldots, u_5, u_6)$, where u_6 corresponds to the six active bits in the left half of the ciphertext. The mask u_6 will be determined by the rest of parts of the mask, as from Corollary 5.6 we can deduce that

$$\widehat{f_2}(u_0, \ldots, u_5, u_6) \neq 0 \Longrightarrow (u_6[i] = 1 \longleftrightarrow u_i \neq 0 \text{ for all } i). \tag{5.50}$$

Furthermore, the following expression for the Walsh coefficient can be deduced:

$$\begin{aligned} &\widehat{f_2}(u_0, \ldots, u_5, \mathrm{ind}(u_0, \ldots, u_5)) = \\ &\quad \widehat{S_8}(u_0, \mathtt{1})\widehat{S_4}(u_1, \mathtt{4})\widehat{S_6}(u_2, \mathtt{1})\widehat{S_2}(u_3, \mathtt{1})\widehat{S_1}(u_4, \mathtt{4})\widehat{S_3}(u_5, \mathtt{4})\widehat{S_5}(\mathrm{ind}(u_0, \ldots, u_5), \mathtt{F}), \end{aligned} \tag{5.51}$$

where $\mathrm{ind}(u_0, \ldots, u_5)[i] = 1 \iff u_i \neq 0$. We also collect the following information about the Walsh spectra of the active Sboxes in the last round:

- $\widehat{S_8}(\,\cdot\,, \mathtt{1})$ has 15 zeros.
- $\widehat{S_6}(\,\cdot\,, \mathtt{1})$ has 17 zeros.
- $\widehat{S_1}(\,\cdot\,, \mathtt{4})$ has 14 zeros.
- $\widehat{S_4}(\,\cdot\,, \mathtt{4})$ has 12 zeros.
- $\widehat{S_2}(\,\cdot\,, \mathtt{1})$ has 20 zeros.
- $\widehat{S_3}(\,\cdot\,, \mathtt{4})$ has 18 zeros.

By adding the number of non-zero coefficients associated to each value in Table 5.7, we conclude that despite being a map defined in $\mathbb{F}_2^{42}$, the Walsh spectrum of f_2 only has around $2^{30.31}$ non-zero coefficients. Furthermore, since $\widehat{S_5}(\mathtt{3F}, \mathtt{F}) = 0$, $u_0, \ldots, u_5$ cannot all be non-zero at the same time. All non-zero coefficients belong to at least one of six vector subspaces of dimension 35. Each subspace $\widetilde{U_i}$ is determined by fixing one $u_i = 0$, which is a six bit condition, as well as the bit condition $u_6[i] = 0$. Since $\widehat{S_5}(\mathtt{3D}, \mathtt{F}) = 0$ we can also remove the subspace $\widetilde{U_1}$.

5.4.2.2 Attack Algorithm and Complexity

Based on the observations we have made on the key recovery map for the attack, we propose the following attack algorithm:

1. **Distillation phase and first set of Walsh transforms.** The non-zero values of the Walsh spectrum of the key recovery map form five affine subspaces, and each subspace will be handled separately.

Table 5.7: Part of the Walsh spectrum of S_5: $\widehat{S_5}(\cdot, \mathtt{F})$.

00	0	08	8	10	−40	18	−8	20	0	28	0	30	8	38	0
01	0	09	−8	11	8	19	−8	21	0	29	0	31	8	39	0
02	−8	0A	0	12	0	1A	0	22	−24	2A	8	32	0	3A	−8
03	−8	0B	0	13	0	1B	0	23	−8	2B	8	33	0	3B	8
04	0	0C	−8	14	0	1C	0	24	0	2C	0	34	0	3C	8
05	8	0D	0	15	8	1D	8	25	−8	2D	−8	35	8	3D	0
06	0	0E	−8	16	0	1E	0	26	0	2E	0	36	0	3E	8
07	−8	0F	0	17	8	1F	−8	27	−8	2F	−8	37	−8	3F	0

The first step in the analysis phase will consist of five pruned Walsh transform whose inputs are restricted to a subspace Y of dimension 40 (determined by the duplicate input bits in the key recovery map) and whose outputs are restricted to subspaces $U_i = \mathbb{F}_2^6 \times \widetilde{U_i}$ of dimension 41. It can be shown that $\dim(Y/(Y \cap U_2^\perp)) = 33$, $\dim(Y/(Y \cap U_0^\perp)) = 35$, and $\dim(Y/(Y \cap U_i^\perp)) = 37$ for $i = 3, 4, 5$.

(a) The distillation phase begins by initialising three arrays g_3, g_4, g_5 of length 2^{37}, one array g_0 of length 2^{35} and one array g_2 of length 2^{33}.

(b) For each plaintext-ciphertext pair, we increment or decrement one position in each of the five arrays according to the values of the appropriate parts of the plaintext and ciphertext and to the value of $P[39] + P[50] + P[56] + C[7] + C[18] + C[24] + C[29]$.

(c) We apply a fast Walsh transform on each of the five arrays.

The time complexity of these steps is around $6N$ memory accesses and $3 \cdot 37 \cdot 2^{37} + 35 \cdot 2^{35} + 33 \cdot 2^{33} \simeq 2^{43.93}$ additions and subtractions.

2. **Multiplying by the Walsh coefficients.** There are $50 \cdot 2^{30.31} \simeq 2^{35.95}$ non-zero coefficients of the Walsh spectrum of the key recovery map in total. They can be easily enumerated by separating them into 32 sets (one for each non-zero value of $\widehat{S5}(\,\cdot\,, \mathtt{F})$) and looking at the non-zero positions in the LATs of (up to) 7 other active Sboxes.

(a) We initialise one array h_2 of length 2^{38}, one array h_0 of length 2^{37}, and three arrays h_3, h_4, h_5 of length 2^{34} for the second stage of Walsh transforms.

(b) For each of the non-zero Walsh coefficients, we retrieve the associated output of the first Walsh transform from one of the g_i (if the coefficient lies in more than one of the U_i, we can choose any one of them). We then multiply it by the coefficient and add or subtract the result to the appropriate position the array h_i.

The time complexity of this step is $7 \cdot 2^{35.95} \simeq 2^{38.76}$ products and the same number of additions. We note that the majority of the products can be carried out using bit shifts.

3. **Second set of Walsh transforms and exhaustive search.** The Walsh transforms in the second set are pruned at the inputs according to the five subspaces U_i, and at the outputs according to a subspace V of dimension 40 which is determined by the key schedule. We can show that $\dim(V/(V \cap U_2^\perp)) = 38$, $\dim(V/(V \cap U_0^\perp)) = 37$, and $\dim(V/(V \cap U_i^\perp)) = 34$ for $i = 3, 4, 5$.

(a) We first perform the standard fast Walsh transform on the five arrays h_0, h_2, h_3, h_4, h_5.

(b) For each of the 2^{40} possible key guesses, we add one coordinate from of each of the five arrays to obtain the experimental correlations. We keep the 2^{24} guesses with the highest correlation, as we aim for an advantage of 16 bits.

(c) For each one of the partial key guesses which were obtained in the previous step, we try all possibilities of the 16 other keybits exhaustively until either the correct key is found or the attack fails.

The time complexity of these steps is $38 \cdot 2^{38} + 37 \cdot 2^{37} + 3 \cdot 34 \cdot 2^{34} + 5 \cdot 2^{40} \simeq 2^{44.41}$ additions/subtractions and 2^{40} trial encryptions.

The data complexity of the attack was once again determined using the model of Blondeau and Nyberg [BN17] which was discussed in Section 2.4. We obtain a 16 bit advantage with 70% probability

with $N = 2^{41.5}$ data. The memory complexity is dominated by the ten large arrays, together require $2^{39.74}$ memory registers of 64 bits. This can be reduced to around $2^{38.75}$ by performing the multiplication by the Walsh coefficients in a way in which the g_i and the h_i do not have to be allocated at the same time.

For the time complexity, we consider that on a modern processor a DES encryption takes 16 clock cycles, a product takes 6 clock cycles, and a memory access or an addition take 1 clock cycle. Multiplying the number of operations of each type by these factors, we obtain

$$\frac{1}{16} \cdot 6 \cdot 2^{41.5} + \frac{1}{16}\left(2^{43.93} + 2^{44.41}\right) + \frac{6}{16} \cdot 2^{38.76} + 2^{40} \simeq 2^{42.13} \text{ DES encryptions.}$$

This attack is, to the best of our knowledge, the best in terms of data complexity. The time complexity is high compared to previous attacks but only if the data generation is excluded. The memory complexity is especially high when compared with previous attacks.

5.4.3 Application to Noekeon

As an additional example, we will describe some linear attacks on 12-round Noekeon [DPVAR00]. This attack was originally presented in [BCF+21] as an application of the binary decision tree key guessing technique. In particular, we made use of the fact that, for some components of the Noekeon Sbox, the output can be determined by querying only three of the Sbox input values. In this chapter, we will show that the key recovery algorithm based on Walsh transform pruning can also detect this behaviour and eventually leads to an equivalent attack algorithm.

5.4.3.1 Noekeon Specification and Basic Linear Attack

Noekeon is a block cipher which was presented by Daemen et al. ([DPVAR00]) to the NESSIE competition. It has a block and key length of 128 bits, and its designers suggest using 16 iterated rounds, which are enumerated from 0 to 15. The designers proposed both *direct key mode* Noekeon and *indirect key mode* Noekeon. In direct key mode, the master key itself is used as the round subkey in every round. In indirect key mode, which is used when related-key attacks are a concern, the working key is obtained by applying Noekeon with an all-zeros key on the master key itself.

The Noekeon state a consists of four 32-bit words, $a[0]$ to $a[3]$. The round subkey or working key is the same for all the rounds. Each round consists of the following transformations:

1. **Round constant addition:** A round constant RC_i is xored to $a[0]$. The round constants are determined using the following LFSR:

$$RC_0 = \texttt{80}, \quad RC_{i+1} = \begin{cases} RC_i \ll 1 & \text{if } RC_i \wedge \texttt{80} = \texttt{0} \\ (RC_i \ll 1) \oplus \texttt{1B} & \text{if } RC_i \wedge \texttt{80} \neq \texttt{0} \end{cases} \tag{5.52}$$

2. **Linear layer:** A keyed linear transformation θ is applied to the state.

$$\begin{aligned}
\theta(a)[0] &= a[0] \oplus a[1] \oplus a[3] \oplus a[1] \lll 8 \oplus a[3] \lll 8 \oplus a[1] \ggg 8 \oplus a[3] \ggg 8 \\
&\quad \oplus k[0] \oplus k[1] \oplus k[3] \oplus k[1] \lll 8 \oplus k[3] \lll 8 \oplus k[1] \ggg 8 \oplus k[3] \ggg 8 \\
\theta(a)[1] &= a[1] \oplus a[0] \oplus a[2] \oplus a[0] \lll 8 \oplus a[2] \lll 8 \oplus a[0] \ggg 8 \oplus a[2] \ggg 8 \\
&\quad \oplus k[1] \\
\theta(a)[2] &= a[2] \oplus a[1] \oplus a[3] \oplus a[1] \lll 8 \oplus a[3] \lll 8 \oplus a[1] \ggg 8 \oplus a[3] \ggg 8 \\
&\quad \oplus k[2] \oplus k[1] \oplus k[3] \oplus k[1] \lll 8 \oplus k[3] \lll 8 \oplus k[1] \ggg 8 \oplus k[3] \ggg 8 \\
\theta(a)[3] &= a[3] \oplus a[0] \oplus a[2] \oplus a[0] \lll 8 \oplus a[2] \lll 8 \oplus a[0] \ggg 8 \oplus a[2] \ggg 8 \\
&\quad \oplus k[3]
\end{aligned}$$

   If we ignore the key addition, the transformation θ is involutive. We can consider the working key is added either before or after θ by transforming it into an equivalent key.

3. **First shift:** A shift operation π_1 is applied to the state:

$$\begin{aligned}
\pi_1(a)[0] &= a[0] & \pi_1(a)[1] &= a[1] \lll 1 \\
\pi_1(a)[2] &= a[2] \lll 5 & \pi_1(a)[3] &= a[3] \lll 2
\end{aligned}$$

Algorithm 17: The NOEKEON block cipher

Input: A plaintext x of 128 bits, a working key K of 128 bits, a number of rounds r.
Output: A 128-bit ciphertext y.

$y \leftarrow x$;
for $i \leftarrow 0$ **to** $r - 1$ **do**
  $y \leftarrow y \oplus RC_i$;
  $y \leftarrow \theta(y, K)$;
  $y \leftarrow \pi_1(y)$;
  $y \leftarrow \gamma(y)$;
  $y \leftarrow \pi_2(y)$;
end
$y \leftarrow y \oplus RC_r$;
$y \leftarrow \theta(y, K)$;
return y;

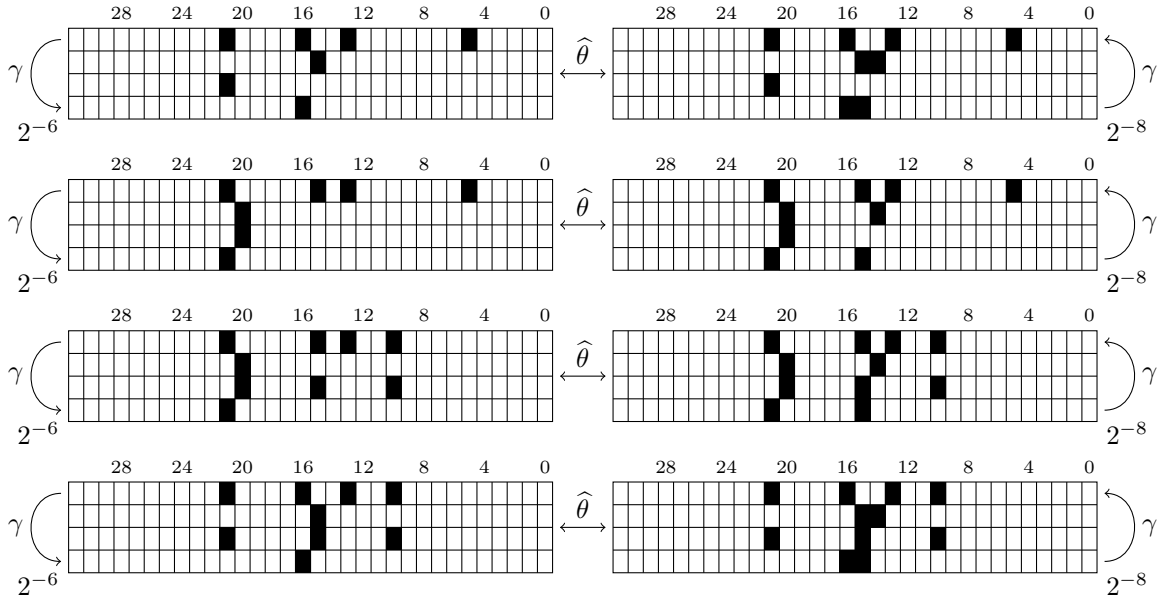


Figure 5.16: Four two-round iterative linear trails of NOEKEON.

4. **Non-linear layer:** A non-linear map γ is applied to the state. It consists of the parallel application of a four-bit involutive Sbox:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	A	2	C	4	8	F	0	5	9	1	E	3	D	B	6

5. **Second shift:** The shift operation π_2, which is the inverse of π_1, is applied to the state.

All of the operations θ and γ are involutive, and that π_2 is the inverse of π_1, which means that each round of decryption can be performed by swapping the round constant addition to after the application of θ. We use the shorthand $\widehat{\theta}$ to denote the involutive map $\pi_1 \circ \theta \circ \pi_2$.

The designers of NOEKEON [DPVAR00] make mention of a nine-round linear trail with correlation 2^{-62} which is obtained from an iterative two-round linear trail with correlation 2^{-14}. This trail can be used as a distinguisher between rounds 1 and 9 to mount a 12-round linear attack with the following key recovery structure:

$$\underbrace{\overbrace{\theta\ \pi_1\ \gamma}^{\text{Round 0}}}_{\text{Key rec.}}\ \overbrace{\pi_2\ \theta\ \pi_1\ \gamma\ \pi_2}^{\text{Round 1}}\ \cdots\ \overbrace{\theta\ \pi_1\ \gamma\ \pi_2}^{\text{Round 9}}\ \overbrace{\theta\ \pi_1}^{\text{Round 10}}\ \underbrace{\gamma\ \pi_2}_{\text{Key rec.}}\ \overbrace{\theta}^{\text{Round 11}}\ \underbrace{\pi_1\ \gamma\ \pi_2}_{\text{Peelback}}$$

Linear approximation

By guessing 24 bits of two transformed working keys, one corresponding to a key addition after $\widehat{\theta}$ in round 0 and the other to a key addition before $\widehat{\theta}$ in round 11. This means a guess of 48 keybits in total. The

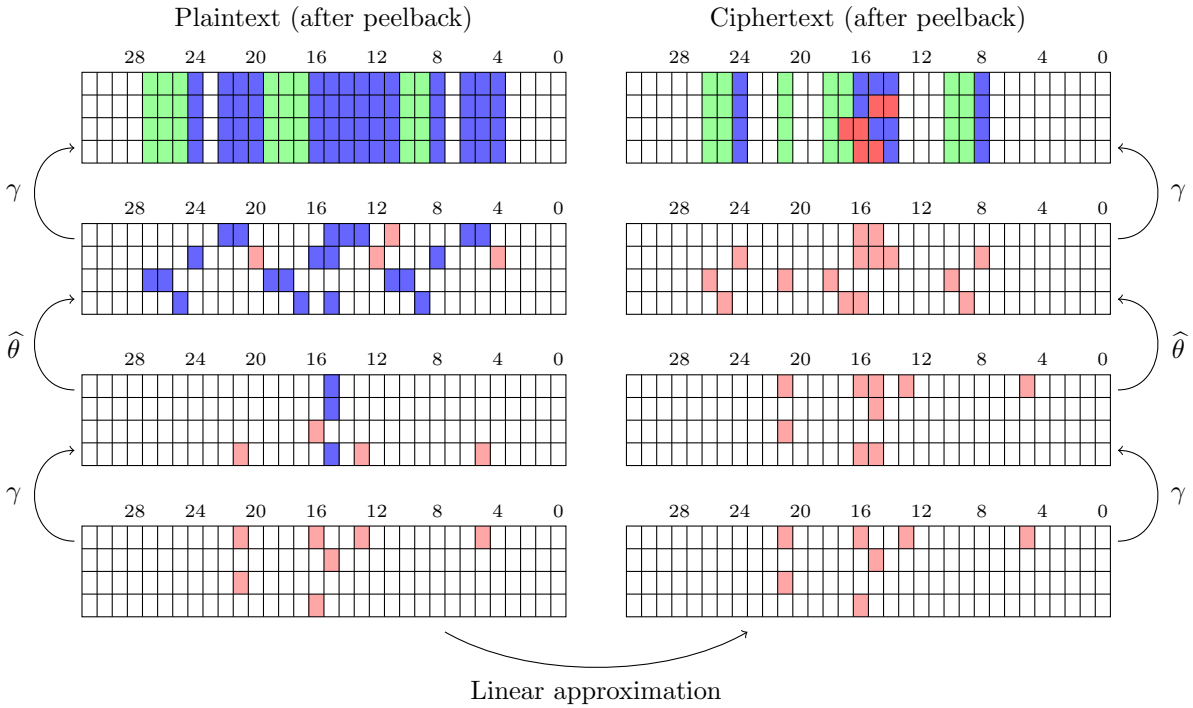


Figure 5.17: Attack on 12-round NOEKEON with 2^{119} data and $2^{124.5}$ time complexity.

data complexity is estimated to be around $2^{62 \cdot 2} = 2^{124}$ known plaintexts. If a distillation table is used as in [Mat94a], the time complexity is $2^{124} + 2^{48 \cdot 2} = 2^{124}$.

5.4.3.2 Improved Attacks on 12-round NOEKEON

Our attacks are also based on iterative two-round linear trails with correlation 2^{-14}. Since all the transformations in a NOEKEON round except for the constant and key additions are invariant under rotation, we can obtain new trails from known ones by rotating the mask and/or swapping both rounds of a trail. Using an MILP model (see Chapter 7), we have identified four families of trails, which are shown in figure 5.16. We thus have $2 \cdot 32 \cdot 4 = 256$ different linear trails we could potentially use in our attacks.

We propose a 12-round attack which is based on a nine-round linear trail which is obtained by extending the first iterative linear trail and has a correlation of 2^{-62}, in an analogous way to the previous linear attack. We make some modifications to this linear trail which will reduce the data complexity to 2^{119}:

- Sbox 15 is removed from the first round of the linear trail. This leads to an input mask which is "straddles" rounds 1 and 2, and increases the correlation by a factor of 2^2.
- In the last round, we substitute the Sbox 15 approximation from $\mathtt{2} \to \mathtt{2}$ to $\mathtt{2} \to \mathtt{b}$, which increases its correlation from 2^{-2} to 2^{-1}.
- Other modifications were made to the Sbox linear approximations in rounds 1 and 14 in order to minimise the number of active keybits in the key recovery.

The correlation of the linear trail increases from 2^{-62} to 2^{-59}. However, in a key recovery attack, we would need to guess 92 key bits in the first round, 4 in the second, and 48 in the last. Even if an FFT approach is used, if no pruning is applied, the time complexity surpasses 2^{144}. We look at the properties of the Sbox:

- In the first and last rounds, whenever we need to compute the output y_2 or y_3 of an Sbox, we notice that it only depends on three bits of the input. In [BCF+21] (see Chapter 6, we used this to reduce the key guess by one bit in each of these Sboxes. However, when looking at the Walsh spectrum of S, these properties appear as a hyperplane of $\mathbb{F}_2^4$ which contains all the non-zero Walsh coefficients. For example, the fact that the value of the input x_3 does not influence the value of y_3 is equivalent to the linear equation $u_3 = 0$ for all non-zero values of $\widehat{S}(u, \mathtt{8})$.

Table 5.8: Sbox properties used in the NOEKEON attack, as described in [BCF+21].

x_2	$x_0 \oplus x_1$	$x_0 \oplus x_3$	y_2
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

x_0	x_1	x_2	y_3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

x_0	x_1	x_3	y_1
0	0	0	?
0	0	1	?
0	1	0	1
0	1	1	?
1	0	0	?
1	0	1	0
1	1	0	0
1	1	1	1

- In [BCF+21] we also used a slightly more complex property for Sbox 15 in the second round. In particular, that if only x_0, x_1 and x_3 are known, y_1 is still determined for half of the possible inputs. If all the plaintexts for which y_2 is not determined are rejected, then the second input can be ignored with regard to the key recovery. This property also appears as a hyperplane of $\mathbb{F}_2^4$ which contains all the non-zero coefficients of a restricted Walsh spectrum of S.

In [BCF+21], a 12-round FWT linear key recovery attack was performed with a reduced key guess of 124 bits. The time complexity was also reduced by using the Walsh transform pruning technique of [FN20] in order to prune the last set of Walsh transforms at the output side. There are three keybits which are active in both the first and second rounds, as well as six last round key bits which can be deduced from the first round. This allowed a time complexity of

$$2^3 \cdot \left(2^{121} + (121-9) \cdot 2^{121-9}\right) \simeq 2^{124.29} \text{ additions.} \tag{5.53}$$

The details of the key recovery are specified in Figure 5.17. Light pink bits represent the masks of the linear approximation. The active bits for the key recovery are coloured in dark blue. In the plaintext and the ciphertext side, we have also coloured the Sboxes with reduced 3-bit key guesses in light green. The six bits of the last round key guess which can be deduced from the first round key guess are painted red.

We must also compare the costs of additions and a 12-round NOEKEON encryptions. A conservative estimate of 128 operations per Sbox layer or key addition and 64 operations per linear layer leads to at least 3840 bit operations for a full encryption. An addition 128-bit integers takes around 256 bit operations. Therefore, its cost is at most one fifteenth of the cost of an encryption. The full time complexity is thus $2^{119} + (1/15) \cdot 2^{124.29} \simeq 2^{120.85}$ encryptions.

Let us now look at the attack from the perspective of the hyperplanes which cover the Walsh spectrum of the key recovery map. For each 4-bit guess of the inner key, the Walsh spectrum of the key recovery map lies in a vector space of dimension 121 of the ambient space of dimension 144. Furthermore, the Walsh spectrum is independent of one of the bits of the inner key guess, so it can be reduced to just three bits. The first Walsh transform of the algorithm has its outputs restricted to a space of dimension 121, and the first step of the pruned algorithm can be incorporated into the distillation phase. The second set of Walsh transforms is limited at the input to a subspace of dimension 121 and at the output to a subspace of dimension 135. The cost of these transforms is $112 \cdot 2^{112}$ additions each, after checking that both subspaces are orthogonal. Since the last step of these pruned transforms consists of mere copying of values, and we are only interested in the highest correlations, we can just find them without expanding the output. The end result is an algorithm with the same complexity as the one we described in [BCF+21], showing that the approach introduced in [Fló22] generalises that of [BCF+21] for linear cryptanalysis.

Both versions of the attack have a data complexity of around 2^{119} known plaintext-ciphertext pairs and a time complexity of $2^{120.85}$ encryptions, which is as far as we know the best on 12-round NOEKEON.

5.5 Conclusion and Open Problems

In this chapter we have described all the generalisations and improvements to the classical algorithm of Collard et al. [CSQ07b] which we proposed in [FN20, Fló22]. There are fundamentally three such improvements:

Table 5.9: Sbox properties used in the NOEKEON attack, in terms of restricted Walsh spectra.

$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(\cdot,4)$	u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$\mathcal{X}=\varnothing$	0	0	0	−8	0	0	0	−8	0	−8	0	0	0	8	0	0
$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(\cdot,8)$	u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$\mathcal{X}=\varnothing$	0	8	0	8	0	8	0	−8	0	0	0	0	0	0	0	0
$\widehat{S}_{S(x)\in\mathbb{F}_2^4\setminus\mathcal{X}}(\cdot,2)$	u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$\mathcal{X}=\varnothing$	0	−4	4	0	0	4	−4	0	0	−4	−4	8	−8	−4	−4	0
	$\mathcal{X}=\{0,1,4,5,8,A,C,E\}$	0	−4	4	0	0	0	0	0	0	−4	−4	8	0	0	0	0

- A generalised version of the algorithm in which the key recovery map can be divided into multiple parts. This version adapts to all the scenarios of key recovery over multiple rounds and allows enough flexibility to describe several variants of the attack. In general, we find that if $|K^O|$ bits of outer key material and $|K^I|$ bits of inner key material have to be guessed, then the time complexity of the analysis phase is $|K^O|2^{|K^O|+|K^I|}$.
- An analysis of the specific case of multiple linear cryptanalysis. In particular, we describe an algorithm which performs partial key guesses for each linear approximation in the distinguisher and then combines the information for the final stage of the attack, using a reduced key guess which can be optimised according to the key schedule.
- A discussion of the application of Walsh transform pruning techniques to the key recovery algorithm which can exploit structure in the data, the key guess, and the cipher construction itself. In particular, we show that under certain conditions the Walsh spectrum of the key recovery map can be split into several affine subspaces of small dimension, thus reducing the time complexity of the key recovery. We also show that rejecting a small proportion of the data can result in a similar advantageous situation.

These improvements are then illustrated with linear attack examples on up to 29-round PRESENT and 12-round NOEKEON, as well as on the DES. In all cases, the new attacks are the best in the literature in terms of number of rounds and data complexity.

Regarding future research, there are several questions which may prove of interest. The most evident continuation is the application of these new techniques to other ciphers and attacks. One case in which we think there may be quite interesting results is that of differential-linear cryptanalysis [LH94, BDK03]. Since the same key guess is used for both ciphertexts in each pair, the desired outputs of the last set of Walsh transforms lie in a vector subspace which is half the dimension of the ambient space. Furthermore, we believe that more application examples may provide further insight into the nature of this algorithm and inspire improved variants.

As it may have become clear in Section 5.4, applying these techniques is a fairly technical and involved task which requires a lot of trial and error as well as intuition so that the appropriate parameters for the attack are chosen. An automatic tool which outputs an optimal key recovery algorithm and its computational complexity within a specific search space given a linear distinguisher of a block cipher could be of great use to the community.

The results of Section 5.3 seem to be much more effective in the case of ciphers with lightweight linear layers, and specifically those whose linear layers are bit permutations. For this reason, we think that it would be of great interest to find generalisations of the algorithm which apply more readily to constructions with denser linear layers. One possible direction would be to study the Walsh spectrum of F in different basis of $\mathbb{CF}_2^n$: although the Walsh spectrum itself may have a lot of non-zero values, in some cases it becomes very sparse after applying a simple linear transformation. Another approach would be to try to implement conditional guessing techniques akin to the ones we presented for other families of attacks in [BCF+21].

Chapter 6

Optimised Key Guessing in Sboxes

This chapter consists of a summary of my joint work with Marek Broll, Federico Canale, Nicolas David, Gregor Leander, María Naya-Plasencia and Yosuke Todo [BCF+21, BCD+22] about the optimisation of key recovery attacks on SPN-type block ciphers using tree descriptions of the Sboxes. This technique is fairly generic and can thus be applied to a wide variety of attack families, as was showcased by all the application examples included in [BCF+21].

Contents

This chapter is organised as follows. Section 6.1 provides a brief example which motivates the techniques which are proposed in the rest of the chapter. Section 6.2 formalises the ideas which were introduced in the example: we define the trees which will be manipulated, we define appropriate efficiency metrics, and establish an algorithm to search for optimal trees. In Section 6.3 we discuss the application of these techniques to the two classical families of attacks: differential and linear cryptanalysis. In Section 6.4, we describe a differential-linear attack on 12-round Serpent [BAK98] which applies this new technique. Finally, we provide some insight into possible future research on the topic.

6.1 Motivation and Example

We will first provide a small example which will highlight the motivation behind our techniques for the improvement of key recovery attacks. The aim is to show that in many cases, we can reduce the total number of bits of information about the key which have to be guessed in order to fully determine some output(s) of an Sbox. This is achieved by means of representing the desired Boolean function as a binary decision diagram or BDD [Bry86].

Let us consider the Sbox used in the block cipher Noekeon [DPVAR00], which corresponds to the nonlinear layer γ of the specification (see Subsection 5.4.3.1). The Sbox is given as a lookup table (reproduced in Table 6.1), while in the original specification it was given as a circuit of logical gates. For the purposes of key recovery, these kinds of representations are not very useful, as they may obfuscate

Table 6.1: The inputs of the NOEKEON Sbox and the most significant output bit $f(x)$.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	A	2	C	4	8	F	0	5	9	1	E	3	D	B	6
x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$f(x)$	0	1	0	1	0	1	1	0	0	1	0	1	0	1	1	0

the way in which the inputs of the Sbox determine the outputs. In particular, it is possible that some bits of the input become irrelevant to some of the outputs after some information about the input is determined. This kind of behaviour is difficult to observe directly on a lookup table.

We will consider the function $f(x) = \langle 8, S(x)\rangle$ corresponding to the the most significant bit of the Sbox output $S(x)$, whose value is also given in Table 6.1. Closer inspection of this table reveals that the first eight values of f are identical to the last eight values. In other words, we find that the value of f is completely independent from the output x_3. This property is sometimes called a *linear structure* of the Boolean function, since we have a vector space $L_0 = \text{span}\{(0,0,0,1)\}$ so that f is constant in all the fibres $x + L_0$:

$$f(x) + f(x + (0,0,0,1)) = 0 \text{ for all } x \in \mathbb{F}_2^4. \tag{6.1}$$

This kind of property constitutes the simplest form of relationship that we want to exploit. However, we can do better if we also consider linear structures which only hold conditionally.

For example, if we know that $x_1 = x_0 = 0$, then the value of $f(x)$ is always 0, and the value of x_2 is irrelevant. The same applies to the case $x_1 = 0$ and $x_0 = 1$, as it implies that $f(x) = 1$. We now just have the case $x_1 = 1$ left to consider. Guessing one extra input bit among x_0, x_2 and x_3 is not sufficient to determine the value of f. However, we note that it is sufficient to determine a linear combination of the remaining bits, as we have $f(x) = x_0 + x_2$. This means that we can just consider two additional cases: if $x_1 = 1$ and $x_0 + x_2 = 0$ we get $f(x) = 0$ and if $x_1 = 1$ and $x_0 + x_2 = 1$, we get $f(x) = 1$. In all cases, we are able to determine the value of f by querying only two parity bits of the input, as opposed to the four we would normally expect.

We can represent these strategies for determining the desired output bit with conditional queries to the inputs using binary trees or binary decision diagrams (BDDs), such as the ones shown in Figure 6.1. In each node, we query one linear combination of the inputs which is given by a mask α, and we continue to either of its two children depending on the value of this linear combination. We start from the root of the tree, and by the time we reach a leaf, we have fully determined the value of the Boolean function. In the classical literature about BDDs [Bry86], the decision variables at each node are often considered to be separate input variables to the Boolean function, while here we consider a slightly more general case in which these decision variables to the nodes can be any linear combination of the inputs.

Although a binary decision diagram is a representation of f which can be used to compute its value for any given input, it is not unique. Indeed, Figure 6.1 shows two different tree representations of the same function f. We note that as cryptanalysts, we are much more interested in the graph at the top, since the average number of queries to the inputs is lower. In particular, the first tree always queries 2 bits of information about the input, while the second tree queries two bits for half of the possible inputs and 3 bits for the other half, which means that it requires 2.5 bit queries on average.

Let us step back and think in terms of a key recovery attack now. During an attack, we may need to compute the output f of the Sbox for a given plaintext and all possible guesses of the key. The input to f takes the form $x + k$, where x is fixed, so we need to guess k. Instead of guessing k directly, we can just consider the values of $\langle \alpha, x + k\rangle$ (equivalent to determining $\langle \alpha, k\rangle$) at the nodes on the path which is taken during the evaluation of $f(x + k)$. In the end, for each fixed x, we find that we must consider a different guess of k for each possible evaluation path through the graph, or, equivalently, one guess for each leaf of the tree. The largest possible number of leaves for an n-bit Sbox is 2^n, which is the number of different key guesses which are made in a traditional key recovery attack. This means that we can reduce the time complexity of the attack if we find a tree which has a significantly smaller number of leaves.

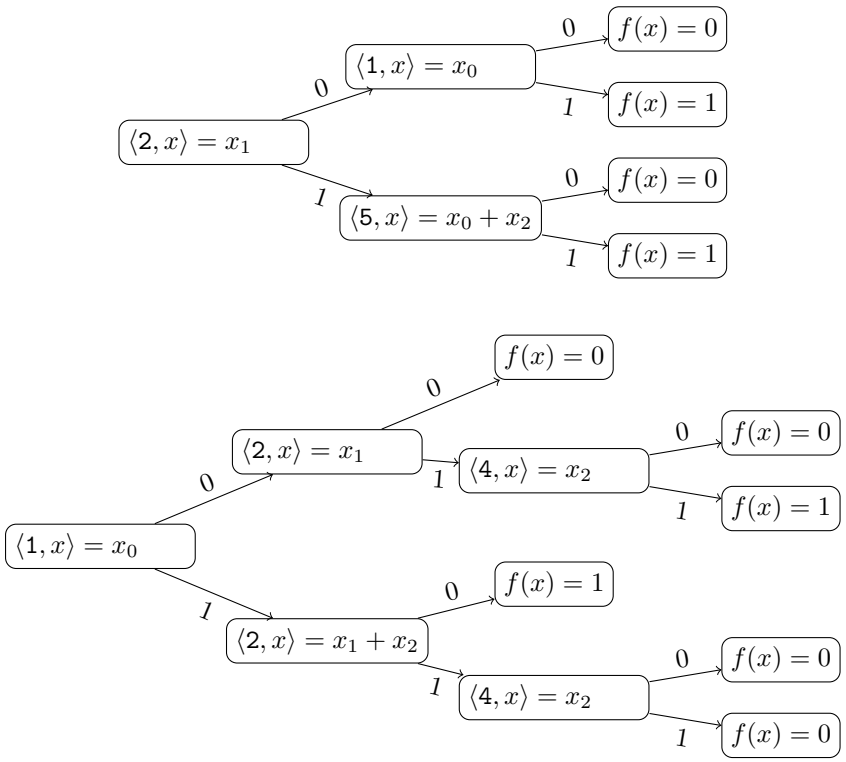


Figure 6.1: Two different binary tree representations of the Boolean function $f = \langle 8, S(\cdot)\rangle$. We can see that one is more efficient than the other, as one requires just two bit guesses for each leaf and the other requires between 2 and 3.

6.2 Decision Tree Representation of Boolean Functions

We will now formalise the ideas which were outlined in the previous example. The objects we will manipulate are binary decision trees which represent (vectorial) Boolean functions. Our aim is to provide a formal definition of these trees, as well as a way to quantify the cost of determining the output in terms of input queries.

Definition 6.1 (Affine binary decision tree). *An* (n, m)-affine decision tree *is any regular binary tree whose inner nodes are labelled with elements of* $\mathbb{F}_2^n$ *and whose leaves (nodes without children) are labelled with elements of* $\mathbb{F}_2^m$*. Given an inner node* v*, we will denote its label by* $v.\text{mask}$*, and its two children by* $v.\text{up}$ *and* $v.\text{down}$*. Given a leaf* l*, we will denote its label by* $l.\text{value}$*. As usual, we will identify a tree and its root node* r *for convenience, and write* $v \in r$ *whenever a* v *is a node of the tree* r*. Given a tree* r*, we will denote the depth of the node* v *as* $\text{depth}(r, v)$*, where* $\text{depth}(r, r) = 0$ *and the children of a node* v *have depth* $\text{depth}(r, v) + 1$*.*

Given any tree which fits the previous definition, we can define an associated vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ which is evaluated by following a path starting at the root node and ending at one of the leaves. More precisely:

Definition 6.2. *Given an* (n, m)*-affine decision tree* r *we can construct an associated vectorial Boolean function* $r : \mathbb{F}_2^n \to \mathbb{F}_2^m$*. Given any* $x \in \mathbb{F}_2^n$ *we define its image by* r*, which we'll denote* $r(x)$*, using the following recursive definition:*

1. *If* r *is a leaf, then* $r(x) = r.\text{value}$*.*
2. *If* r *is an inner node and* $\langle r.\text{mask}, x\rangle = 0$*,* $r(x) = r.\text{up}(x)$*.*
3. *If* r *is an inner node and* $\langle r.\text{mask}, x\rangle = 1$*,* $r(x) = r.\text{down}(x)$*.*

Given $f \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$*, if* $r(x) = f(x)$ *for all* x*, we say that* r *is an evaluation tree for* f*, and write* $r \equiv f$*. We say two trees* r, s *are* equivalent *and write* $r \equiv s$ *if* $r(x) = s(x)$ *for all* $x \in \mathbb{F}_2^n$*.*

For any given Boolean function f, a multitude of different evaluation trees exist, as we saw in Figure 6.1. This means that we must devise some way to quantifiably measure the relative effectiveness of

different evaluation trees of the same function. Furthermore, we wish to come up with algorithms which will find optimal evaluation trees for these metrics given any input vectorial Boolean function.

We start with the first problem, and initially propose the following intuitive definition.

Definition 6.3 (Expected input queries). *Let r be an affine binary decision tree. We define its* average *or* expected number of input queries *as the expected value of the depth of the leaf reached in the evaluation of r when the input x is uniformly distributed in $\mathbb{F}_2^n$. If $l(x)$ denotes the leaf at the end of the path through the tree used in the evaluation of $x \in \mathbb{F}_2^n$, we have:*

$$\text{avgdepth}(r) = \text{Exp}_x\left(\text{depth}(r, l(x))\right) = \frac{1}{2^n}\sum_{x\in\mathbb{F}_2^n} \text{depth}(r, l(x)). \tag{6.2}$$

This quantity measures the number of times, on average, that we will have to query a parity bit of the input x before the value of f is fully determined. If we evaluate a Boolean function directly from a lookup table, this number of queries is equal to the total number of input bits n. Please note that the expected number of queries is defined as the average over the possible inputs, and not over the leaves of the tree. Indeed, the depth of shallower leaves contributes with a heavier weight, as these leaves are reached by a larger proportion of the possible inputs than the deeper leaves.

In order to simplify these definitions, we can assume that the affine binary decision trees don't have any redundant labels. More precisely, we will consider that v.mask forms a linearly independent set with all the masks from its successive ancestor nodes. Indeed, if v.mask was a linear combination of previously queried masks, then the value $\langle v.\text{mask}, x\rangle$ would be determined and all the inputs would go to the same child node. Assuming that the tree has no such redundant nodes, a leaf of depth d is used in the evaluation of precisely 2^{n-d} inputs, and therefore contributes to the average depth with a weight of 2^{-d}. This leads to the following alternative formula for the expected number of queries:

$$\text{avgdepth}(r) = \sum_{\substack{l\in r\\ l \text{ leaf}}} \text{depth}(r, l(x))2^{-\text{depth}(r,l(x))}. \tag{6.3}$$

However, while avgdepth(r) can be useful to represent, for example, the computational cost of evaluating a function using the tree r, it is not very helpful for the purposes of cryptanalysis. In the case of a key recovery attack, we are interested in the number of different guesses of the key that we will have to perform, independently of the number of bits of the key which are determined in each of the individual guesses. For this reason, we propose the following alternative metric:

Definition 6.4 (Number of leaves). *Given an affine binary decision tree r, we denote its total* number of leaves *by:*

$$\text{nbleaves}(r) = |l \in r : l \text{ is a leaf}| \tag{6.4}$$

The number of leaves of an evaluation tree of a function thus denotes the number of possible different paths which can be taken when evaluating the function, and its value for a standard evaluation is 2^n. In the example of the previous section, in Figure 6.1, we see that for the top tree r_1 we have nbleaves$(r_1) = 4$ and for the bottom tree r_2 we have nbleaves$(r_2) = 6$.

Finally, we will consider a third metric, which measures the total dimension of the space of parity bits which may be queried in the evaluation of any input:

Definition 6.5 (Domain size). *The* actual linear domain *of an affine binary decision tree r is the space spanned by all inner node labels:*

$$\text{Dom}(r) = \text{span}\left\{v.\text{mask. } v \text{ is an inner node of } r\right\}. \tag{6.5}$$

We call its dimension the domain size *of r:*

$$\text{domsize}(r) = \dim\left(\text{Dom}(r)\right). \tag{6.6}$$

For the standard evaluation of a Boolean function, the domain size is equal to n, as the actual linear domain is the full space $\mathbb{F}_2^n$. In both trees from Figure 6.1, we find that we obtain the same actual linear domain Dom(r) despite both of the trees having different labels, and it consists of all the vectors α of the form $\alpha = (0, *, *, *)$. In particular, in both cases the domain size is equal to 3, instead of the expected 4.

We can also look at these metrics from the perspective of the associated Boolean function instead of the trees. Given a Boolean function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$, we wish to find affine binary decision diagrams which minimise these cost metrics. For this reason, we define the optimal average depth, the optimal number of leaves, and the optimal domain size:

Definition 6.6. *Let $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a vectorial Boolean function. We define its* optimal average depth*, its* optimal number of leaves*, and its* optimal domain size *as:*

$$\text{avgdopt}(f) = \min_{\substack{r \ tree \\ r \equiv f}} \text{avgdepth}(r) \tag{6.7}$$

$$\text{nblopt}(f) = \min_{\substack{r \ tree \\ r \equiv f}} \text{nbleaves}(r) \tag{6.8}$$

$$\text{domopt}(f) = \min_{\substack{r \ tree \\ r \equiv f}} \text{domsize}(r) \tag{6.9}$$

Given the target Boolean function f, we say that any tree which achieves minimal average depth is avgdepth$-$*minimal, any tree which minimises the number of leaves is called* nbleaves$-$*minimal, and any tree which optimises the domain size is called* domsize$-$*minimal.*

We are interested in finding trees which optimise at least one of these metrics, focusing mainly on the latter two as they are the most interesting to cryptanalysis. As we will see in the next section, we do not need to consider the two problems separately, as optimising nbleaves will also optimise domsize.

6.2.1 Relationship between the Complexity Metrics

We will now try to understand the previous metrics in more depth, so that we can later explain the relationships between them. The first step will be to describe exactly which input values $x \in \mathbb{F}_2^n$ will end up evaluated in the same leaf l of the tree r when evaluating the associated Boolean function. As the evaluation of r for a given input x consists of computing a sequence of inner products $\langle r.\text{mask}, x\rangle$ along the path from the root to a leaf, the inputs x which end up in the same leaf are characterised by the specific values of these inner products. Given a node N of a tree r, we denote by $D(N)$ the vector space spanned by the set of all labels on the path from the root to that node, excluding N itself.

$$D(N) = \text{span}\,\{v.\text{mask} : v \text{ is on the path from } r \text{ to } N, v \neq N\} \tag{6.10}$$

In other words, $D(N)$ tells us which inner products have been evaluated in order to reach it. Since evaluating all the inner products on the path also determines the values of their linear combinations, we consider the span of these labels. We will also denote by $N.\text{space}$ the set of all inputs $x \in \mathbb{F}_2^n$ such that the path through the diagram taken when evaluating $r(x)$ contains N. We can describe $N.\text{space}$ as an affine subspace of the form $N.\text{space} = V(N) + a(N)$, where $V(N) \subseteq \mathbb{F}_2^n$ is a vector subspace and $a(N) \in \mathbb{F}_2^n$ is the shift from the origin. We note that, since $V(N)$ consists of all vectors x such that $\langle \alpha, x\rangle = 0$ for all $\alpha \in D(N)$, the relationship $V(N) = D(N)^\perp$ holds. This leads to an alternative interpretation of the actual linear domain:

Lemma 6.7. *Let r be an affine binary decision tree. We have*

$$\text{Dom}(r)^\perp = \bigcap_{N \in r} V(N) = \bigcap_{\substack{N \in r \\ N \ leaf}} V(N). \tag{6.11}$$

Proof. Let's rewrite $\text{Dom}(r)$ in terms of the grouped labels $D(N)$:

$$\text{Dom}(r) = \text{span}\left\{\bigcup_{N \in r} D(N)\right\} = \sum_{N \in r} D(N),$$

and considering the orthogonal spaces we get

$$\text{Dom}(r)^\perp = \bigcap_{N \in r} D(N)^\perp = \bigcap_{N \in r} V(N).$$

Since $D(N_2) \subseteq D(N_1)$ whenever N_2 is a descendant of N_1, we deduce that $V(L) \subseteq V(N)$ if L is a leaf and N is one of its ancestors. This means that we can restrict the intersection to just the leaves of the tree. □

We can also study the actual linear domain in terms of 0-*linear structures* [Eve87] (please note that the definition which will be used here is slightly different from the original). We can think of $0-$linear structures as truncated differentials which lead to a zero output difference with probability one:

Definition 6.8 (Linear Structures). *The set of* $0-$*linear structures of* $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *is*

$$\mathrm{LS}_0(f) = \{\Delta : \forall x \in \mathbb{F}_2^n,\ f(x) + f(x+\Delta) = 0\}. \tag{6.12}$$

It is clear from the definition that LS_0 is always a vector subspace of $\mathbb{F}_2^n$.

Let us now return to the definition of $\mathrm{Dom}(r)$, and let $\Delta \in \mathrm{Dom}(r)^\perp$ be a vector orthogonal to the actual linear domain. Given two inputs $x, y \in \mathbb{F}_2^n$ so that $x + y = \Delta$, we note that at any node N in the evaluation tree, we will have $\langle N.\mathrm{mask}, y\rangle = \langle N.\mathrm{mask}, x\rangle + \langle N.\mathrm{mask}, \Delta\rangle = \langle v.\mathrm{mask}, x\rangle$ since Δ is orthogonal to v.mask. In other words, we have $x + y \in \bigcap_{N\in r} V(N)$, and Δ is a $0-$linear structure of r. This not only means that $r(x) = r(y)$, but also that x and y will follow the same path through the tree during evaluation.

Lemma 6.9. *For any affine binary decision tree* r *we have*

$$\mathrm{Dom}(r)^\perp \subseteq \mathrm{LS}_0. \tag{6.13}$$

The space of $0-$linear structures LS_0 is an intrinsic property of Boolean functions and in particular is independent of which decision tree we use to evaluate them. As a consequence, we obtain a simple lower bound for the optimal dimension of the actual linear domain $\mathrm{domopt}(f)$ of any function f. Furthermore, this bound is tight:

Theorem 6.10. *Let* $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *be any vectorial Boolean function, and let* r *be an* $(n, m)-$*affine decision tree of* f *which is optimal with respect to* nbleaves*. Then the following equality holds*

$$\mathrm{LS}_0(f) = \mathrm{Dom}(r)^\perp. \tag{6.14}$$

As a consequence, we have

$$\mathrm{domopt}(f) = n - \dim\left(\mathrm{LS}_0(f)\right) = \mathrm{domsize}(r). \tag{6.15}$$

Before delving into the proof for this result, which will occupy the rest of this subsection, we will explain its consequences. First, it guarantees that any tree which is optimal with respect to the number of leaves is actually also optimal with respect to the actual domain size. Second, it shows that the lower bound on the actual domain size given by the $0-$linear structures of f can always be achieved by an evaluation tree. Finally, we can use it to optimise the search algorithms for optimal trees, as we can reduce the search for optimal trees of f modulo $\mathrm{LS}_0(f)$ to obtain a smaller search space which requires less computational cost.

Finally, we would like to reproduce the following result from [BCF+21], which shows that the optimal complexity metrics for a given Boolean function f are invariant under affine equivalence:

Theorem 6.11. *Let* r *be an* $(n, m)-$*affine decision tree which evaluates a function* $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$*, let* $\pi : \mathbb{F}_2^m \to \mathbb{F}_2^m$ *be a permutation and* $B : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *a bijective affine map. We can construct a tree* s *which evaluates* $g = \pi \circ f \circ B$ *which has the same complexity parameters* avgdepth*,* nbleaves*, and* domsize*.*

Corollary 6.12. *The parameters* $\mathrm{avgdopt}(f)$*,* $\mathrm{nblopt}(f)$ $\mathrm{domopt}(f)$ *are all invariant under affine equivalence of vectorial Boolean functions.*

The proof of these results can be found in [BCF+21].

6.2.1.1 Proof of Theorem 6.10

The proof of the theorem is based on the idea that, given an evaluation tree r which has some node whose label belongs to a $0-$linear structure, we can remove this node from the tree without altering the function it evaluates, thus reducing the number of leaves of the tree. This means that any evaluation tree which uses input masks which are in LS_0 will not be optimal. We start by proving the following technical result:

Lemma 6.13. *If* v *is a node in an affine binary decision tree,* $v.\mathrm{left.space} = V + a$*,* $v.\mathrm{right.space} = V + b$ *and* $a + b \in \mathrm{LS}_0$*, then we can swap* v *for* $\mathrm{argmin}_{t\in\{v.\mathrm{right}, v.\mathrm{left}\}}\mathrm{nbleaves}(t)$ *and update the labels to obtain a tree which evaluates to the same Boolean function but has a smaller* nbleaves*.*

Proof. We first note that given the spaces v.space of all the nodes in the evaluation tree, we can easily determine the labels v.mask of each node. As a result, we can think of the tree in terms of the subspaces which lead to each branch instead of the linear mask at each node. We also can assume, without loss of

generality, that nbleaves(v.left) $\leq$ nbleaves(v.right). Since $a+b \in \mathrm{LS}_0$, we know that $f(x) = f(x+a+b)$ for all $x \in V + a$. This means that we can swap v.right for a copy of v.left with the subspaces associated to every node shifted by b. Finally, we note that it is possible to substitute v for a copy of v.left where the subspaces w.space $= W + c$ for all $w \in v$.left are swapped for span $\{W, a+b\} + c$. If $x \in V + a$, the path taken in the new tree will lead to the same result as v.left because $W + c \subseteq \mathrm{span}\{W, a+b\} + c$ for all the pertinent W. Similarly, for all $x \in V + b$ the path taken is equivalent to the one taken in v.right. Since we have effectively removed the branch v.right from the tree, we can also conclude that the total number of leaves has diminished. □

This auxiliary lemma has the following consequence:

Lemma 6.14. *For every node v of an* nbleaves$-$*optimal tree r we have* $\mathrm{LS}_0 \subseteq V(v)$.

Proof. We know from the previous lemma that, if two sister nodes of r have associated spaces which differ by an element of LS_0, then we can transform the tree into an equivalent one which has a smaller number of leaves, so r cannot be nbleaves-optimal. Let us now consider any $\Delta \in \mathrm{LS}_0$, and any node N, and try to show that $\Delta \in V = V(N)$. We will proceed by contradiction, so let us assume that $\Delta \notin V$. This means that $(\Delta + N.\mathrm{space}) \cap N.\mathrm{space} = \varnothing$, because V is the director subspace of N.space. There must therefore be some node further up the tree where the evaluation paths of N.space and $\Delta + N$.space separate. That is, there exists some node v so that N.space $\subseteq v$.space and N.space $+ \Delta \subseteq v$.space, but where, without loss of generality, N.space $\subseteq v$.left.space and $\Delta + N$.space $\subseteq v$.right.space. However, this would mean that v.right.space $= U + a$, v.left.space $= U + b$, $a + b = \Delta \in \mathrm{LS}_0$, which contradicts optimality according to the previous lemma. □

This result concludes the proof of Theorem 6.10.

6.2.2 Tree Search Algorithm

Let us now consider the problem of describing algorithms which search for optimal trees for a given vectorial Boolean function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$. On paper, we can just define a recursive algorithm which goes through all possible values for the root label $\alpha = r$.mask and then calls the same algorithm to construct optimal trees for r.left and r.right, which will model the two parts of the function $f|_{\langle \alpha, x\rangle = 0}$ and $f|_{\langle \alpha, x\rangle = 1}$, whose director subspaces are isomorphic to $\mathbb{F}_2^{n-1}$. This recurrent step is iterated until the resulting parts of the function are constant. Since we are only interested in finding optimal trees, we can apply several optimisations which can greatly reduce the space of the exhaustive search. These improvements mainly consist of removing equivalent trees from the search (that is, trees which will share the same properties) and to perform early abort checks so that several bad trees can be discarded at a time.

- We already know that when building a tree and choosing the mask labels for each node, we can always ignore all the possible masks which are linear combinations of masks which appear earlier in the evaluation path, as these masks provide no additional information about the input. Furthermore, two labels which only differ from each other by a linear combination of previous masks will divide the inputs in the same two subspaces, which means that we only have to consider one such label. Formally, we consider that the label for the node v can is chosen in the quotient space $\mathbb{F}_2^n / D(v)$. This quotient space is parametrised using a complement space of $D(v)$. The algorithm including this optimisation is written in pseudocode as Algorithm 18.
- We can also apply a simple early abort technique: if a subtree of depth d has more than 2^{n-d} leaves, all trees containing it can be removed from the search, as they are necessarily redundant and cannot be nbleaves$-$optimal. Furthermore, we can provide an upper bound for the desired value of nbleaves and reject all subtrees which will unavoidably result in trees with a larger number of leaves.
- If f has non-trivial $0-$linear structures, we can reduce the search problem modulo LS_0. Indeed, we can search for trees for the function $g : \mathbb{F}_2^n / \mathrm{LS}_0 \longrightarrow \mathbb{F}_2^m$ with $g(x + \mathrm{LS}_0) = f(x)$. This optimisation also applies to all the recursive calls to the subtree routine.

We will finally briefly discuss the performance of this algorithm. On a standard PC, we were able to successfully run experiments on randomly-generated Boolean functions on $\mathbb{F}_2^n$ with n as large as 7 after implementing all the optimisations which are mentioned above. In the case of dimension 4 it was actually possible to run the algorithm without optimisations, thus generating all the trees and allowing us to filter them afterwards. The average running times for the optimised algorithm on a 2.3GHz desktop CPU are

Algorithm 18: Listing all evaluation trees of a Boolean function

Input: $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$.
Output: A list of trees for f.
return ListTrees$(f, \mathbb{F}_2^n, \mathbb{F}_2^n)$;
Procedure ListTrees(f, X, Y) **is**
  Input: A Boolean function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$, two affine subpaces $X, Y \subseteq \mathbb{F}_2^n$,
    $X = U + c \subseteq Y = V + c$, where U, V are vector subspaces and $c \in \mathbb{F}_2^n$.
  Output: A list of trees for $f|_X$.
  $L \leftarrow \{\}$;
  if $f|_X$ *is constant* **then**
    Add a tree consisting of a single leaf l with label l.value $= f(c)$ to L;
  else
    Obtain $W \subseteq \mathbb{F}_2^n$ such that $W \oplus U^\perp = V$;
    forall $\alpha \in W \setminus \{0\}$ **do**
      $U_0 \leftarrow \{x \in U : \langle \alpha, x \rangle = 0\}$;
      Choose $c' \in U \setminus U_0$;
      $b \leftarrow \langle c, \alpha \rangle$;
      $X_b \leftarrow U_0 + c$; $X_{1+b} \leftarrow U_0 + c' + c$;
      Initialize a tree with root r and r.mask $= \alpha$;
      $L_{\text{left}} \leftarrow$ ListTrees(f, X_0, X); $L_{\text{right}} \leftarrow$ ListTrees(f, X_1, X);
      forall r *tree* : $(r.\text{left}, r.\text{right}) \in L_{\text{left}} \times L_{\text{right}}$ **do** Add r to L;
    end
  end
  return L;
end

Table 6.2: Average running times for the tree search depending on the input dimension.

n	4	5	6	7	8
Runtime	4ms	190ms	21s	1.7h	<3 weeks (estimated)

summarised in Table 6.2. In the case of 8-bit functions, which is of great interest to the field due to the prevalence of 8-bit Sboxes, we estimate that the running time on such a standard PC should be of a few weeks at most. An implementation of this algorithm, which was kindly created by Marek Broll, can be found at `https://github.com/rub-hgi/ConditionsLib`.

6.3 Application to Key Recovery Attacks

This section provides some general ideas as to how the tree technique which has been explained in the previous pages can be applied to several common cryptanalysis scenarios. In particular, we will study two important cases in key recovery attacks: generating pairs with the appropriate internal input difference for differential cryptanalysis, and linear key recovery attacks.

These two examples are not exhaustive, but we consider that they are quite illustrative and behave in a significantly different way from other attacks. Indeed, in attacks where the aim is to determine some parts of the state and where the plaintext-ciphertext pairs can be analysed independently, the technique can be applied directly without any adaptations. For example, in a meet-in-the-middle attack, since the objective is to determine some internal bits of the cipher for a single plaintext, the tree technique is a natural fit. Indeed, using descriptions of the truncated Sbox outputs as affine binary decision trees, we obtain an efficient partitioning of the keyspace which reduces the overall size of the search space from $2^{|k|}$ (where k is the required key guess) to the product of the number of leaves of the trees for each Sbox.

In cases in which the key addition is not performed on the full state, such as that of GIFT [BPP+17], we can use custom cost metrics for the trees.

6.3.1 Differential Cryptanalysis

The application of the tree technique to differential cryptanalysis and other related attacks is a bit more complicated. In addition to determining the values of some bits, we sometimes have to determine a

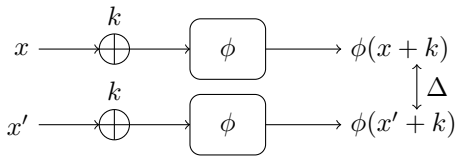


Figure 6.2: Finding plaintext pairs which will produce the desired internal input difference.

difference in some part of an internal state. This means that we have to carefully consider exactly which properties of the state we want to determine, and find optimal trees which describe these properties efficiently. We will consider several situations which might appear in this kind of attack.

6.3.1.1 Constructing Differential Pairs

We consider a differential attack which features key recovery on the first round of a key-alternating block cipher. We wish to create plaintext pairs (x, x') for which the active Sboxes in the first round ϕ have the appropriate output differences Δ so that the second round will receive the correct input difference for a differential distinguisher (see Figure 6.2):

$$\phi(x+k) + \phi(x'+k) = \Delta. \tag{6.16}$$

The first version of this problem we will consider is that of completing a pair. Given x, we want to find x' with the smallest possible guess of k so that we can guarantee that the difference Δ holds. We note that

$$x' = \phi^{-1}(\phi(x+k) + \Delta) + k, \tag{6.17}$$

which suggests defining the function $f_\Delta : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ given by

$$f_\Delta(y) = \phi^{-1}(\phi(y) + \Delta). \tag{6.18}$$

Given a binary decision tree for f_Δ, we can obtain the value of $x' + k$ from a reduced number of bits of $x + k$. Since x is already known, we just need to guess some bits of k. These guessed bits of k can also be used to obtain some parity bits of x' by xoring.

This technique can still miss some "evidently useful" relations, however. For example, if there is a probability 1 differential $\delta \mapsto \Delta$ through ϕ, we have $f_\Delta(y) = y + \delta$. In this case, the tree analysis suggests that we need to guess all the bits of the key, when key-guessing is actually unnecessary since $x' = x + \delta$. In general, computing the value of an expression of the form $\langle \alpha, (x'+k) + (x+k) \rangle$ doesn't require any key guesses. We can find useful properties of this type by studying the trees of the function

$$F_\Delta(y) = f_\Delta(y) + y. \tag{6.19}$$

These trees provide "direct" information about x' (as x is known) where the only required bits of k are those used in the binary decision diagram, and it also detects any information about x' which can be obtained without any key guesses. The cost of the complete guessing is the number of leaves of the tree.

The function F_Δ can also be used in the preliminary sieving of a list of pairs (x, x'). Indeed, if $\delta = x + x'$ is not in the image of this map, then this pair can never produce the correct input difference Δ independently of the key, and can be discarded. The same technique can be used to perform more accurate sieving on the ciphertext side.

Finally, we consider the version of the problem in which we are given a pair (x, x') so that $\delta = x + x' \in F_\Delta(\mathbb{F}_2^n)$ is known, and we want to find a key guess k under which the output difference Δ will hold. We note that under these constraints, the valid inputs which produce the difference Δ are given by the condition $x + k \in F_\Delta^{-1}(\delta)$. In fact, upon closer inspection, we notice that $|F_\Delta^{-1}(\delta)|/2^n$ is the differential probability $\mathrm{DP}(\delta \overset{\phi}{\mapsto} \Delta)$.

We define the function $g_\Delta^\delta : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ given by the property

$$g_\Delta^\delta(x) = 0 \quad \Longleftrightarrow \quad x \in F_\Delta^{-1}(\delta). \tag{6.20}$$

In other words, $g_\Delta^\delta(x) = 0$ if and only if $\phi(x) + \phi(x+\delta) = \Delta$. We can once again compute a tree description of g_Δ^δ so that we can compute its value with as little information on k as possible (again, we will need nblopt different guesses). This can be applied to each valid value of δ, thus diminishing the average cost of key guessing over all $\delta \in \mathbb{F}_2^n$. It can also be applied when the difference Δ is not completely fixed, as given a truncated difference $\overline{\Delta}$ we can still define

$$g_{\overline{\Delta}}^\delta(x) = 0 \quad \Longleftrightarrow \quad \phi(x) + \phi(x+\delta) \in \overline{\Delta}. \tag{6.21}$$

6.3.1.2 Determining Input Differences over Two Rounds

We will next consider the case of attacks featuring two or more consecutive rounds of key guessing. In this case, in addition to determining some of the differences at the output of the first round, we will also require some output values in order to guarantee the appropriate output differences in the second round. In this case, we first consider a guessing tree which determines the output difference of the first round as described in the previous subsection, and then we can use another tree to determine the output values. In the case of the second tree, we can limit the search space to trees which use the bits which are already determined as masks for the first nodes.

Another improvement which can be applied in this kind of situation is what we have called *key absorption*. The idea is to use the fact that binary decision trees often express the value of the function as a linear combination of the inputs (for example, given two sister leaves with a shared parent p, the value of the function is either $\langle p.\text{mask}, x\rangle$ or $1+\langle p.\text{mask}, x\rangle$. This means that, for the specific x for which evaluation passes through this node, one bit of the key guess will appear linearly on the value of the function. This bit can be carried over to the next round, and will be xored to some other keybit. Instead of guessing the two bits separately, we can instead guess their XOR.

We consider two consecutive rounds of key guessing, and will denote by k the round key of the first round and by κ the round key of the second one. Given a pair of plaintexts (x, x'), the first operation is the addition of the first round subkey k, which gives us the pair $(x \oplus k, x' \oplus k)$ at the input of the first Sbox layer. We will denote by (y, y') the output of this Sbox layer. Some linear transformation L is applied next, $(z, z') = (L(z), L(z'))$. The input to the second Sbox layer is thus $(z + \kappa, z' + \kappa)$, and we will denote by (t, t') the output of the second Sbox layer. We assume that in order to determine the output difference to the second Sbox layer $t + t'$, we must know the value of the input bit $z_0 + \kappa_0$. For some mask α, we have

$$z_0 + \kappa_0 = L_0(y) + \kappa_0 = \langle \alpha, y\rangle + \kappa_0.$$

Normally, we would guess the keybit κ_0 directly and compute $\langle \alpha, y\rangle$ using a tree description for the first Sbox layer. However, there will be some sets of inputs (x, x') for which this value $\langle \alpha, y\rangle$ is equal (up to a constant) to a linear combination $\langle \gamma, x + k\rangle$, which means that we would have

$$z_0 + \kappa_0 = \langle \gamma, x + k\rangle + \kappa_0.$$

In this case, guessing the linear combination $\langle \gamma, k\rangle + \kappa_0$ allows us to determine the value of $z_0 + \kappa_0$ with a key guess which is effectively one less bit than if we considered both rounds separately. We cannot absorb the same keybit twice, so it is important to keep track of all the bits which have been absorbed in the second round so that we make no mistakes.

6.3.2 Linear Cryptanalysis

Although this generalised tree-based approach can effectively reduce the time complexity of many key-recovery attack families, there are cases in which other accelerations may provide better results, and a method must be picked. Indeed, in the case of linear cryptanalysis, a set of tools based on the fast Walsh transform exists, as discussed in detail in Chapter 5. The aforementioned chapter also discusses how the specific properties of the key recovery map can be exploited to the advantage of the attacker, in particular, zeroes in the Walsh spectrum of the Sboxes can be leveraged in order to avoid unnecessary computations. This set of techniques, as opposed to binary decision diagrams, is specifically tailored to linear cryptanalysis. However, we will briefly discuss the use of the tree description in the case of linear attacks for the sake of completeness.

We start from the classical Matsui implementation of Algorithm 2 which, for a set of N known plaintext-ciphertext pairs and a key guess of $|k|$ bits, has a time complexity of $O\left(N2^{|k|}\right)$ operations. We can construct a tree for the S-box layer which has a number of leaves nblopt $\leq 2^{|k|}$ which is smaller than the total number of key guesses. For each plaintext, the total space of key guesses is partitioned into nblopt regions, each one of which corresponds to a different leaf of the tree. These regions are different for each plaintext. We thus have to keep a separate set of nblopt key guess counters for each of these plaintext groupings. Once all the data has been processed, we can combine these different tables into full guesses of k bits of the key, until the complete guess with the highest counter can be located. This means we can reduce the time complexity of this kind of attack to $O\left(N \cdot \text{nblopt}\right) + O\left(2^{|k|}\right)$.

We next consider the linear attack with separate distillation and analysis phases with time complexity $O(N) + O\left(2^{2|k|}\right)$ as in [Mat94a]. We can construct some binary decision trees for the Sboxes, but we find that the distilled table still has to be of size 2^{domopt}, even if the number of key guesses for each element of the table can be reduced to just nblopt. This is because the distillation table must contain the necessary

Table 6.3: Summary of (working) 12-round attacks on 256-bit Serpent. Some of the complexities of previous attacks have been adjusted, see 6.4.1.

Rounds	Type	Complexity			Source
		Data	Time	Memory	
$0 \to 11$	Multidimensional linear	$2^{125.8}$	$2^{253.8}$	$2^{125.8}$	[NWW11, MC13]
$0 \to 11$	Multidimensional linear	$2^{125.8}$	2^{242}	2^{236}	[NWW11, MC13]
$4 \to 15$	Differential-linear	2^{127}	2^{251}	2^{127}	[LLL21]
$0 \to 11$	Differential-linear	$2^{127.92}$	$2^{233.55}$	$2^{127.92}$	[BCD+22], this
$0 \to 11$	Differential-linear	$2^{125.74}$	$2^{236.91}$	$2^{125.74}$	[BCD+22], this
$0 \to 11$	Differential-linear	$2^{118.40}$	$2^{242.93}$	$2^{118.40}$	[BCD+22], this

Table 6.4: The Serpent Sboxes.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x)$	3	8	F	1	A	6	5	B	E	D	4	2	7	0	9	C
$S_1(x)$	F	C	2	7	9	0	5	A	1	B	E	8	6	D	3	4
$S_2(x)$	8	6	7	9	3	C	A	F	D	1	E	4	0	B	5	2
$S_3(x)$	0	F	B	8	C	9	6	3	D	1	2	4	A	7	5	E
$S_4(x)$	1	F	8	3	C	0	B	6	2	5	4	A	9	E	7	D
$S_5(x)$	F	5	2	B	4	A	9	C	0	3	E	8	D	6	7	1
$S_6(x)$	7	2	C	5	8	4	6	B	E	9	1	F	D	3	A	0
$S_7(x)$	1	D	F	0	E	8	2	B	7	4	C	A	9	3	5	6

plaintext information for every possible key guess. The best time complexity reduction we can achieve on this attack algorithm is thus $O(N) + O\left(\text{nblopt} \cdot 2^{\text{domopt}}\right) + O\left(2^{|k|}\right)$.

Regarding the FFT linear key recovery attack based on the fast Walsh transform, we can reduce the time complexity to $O\left(\text{domopt} \cdot 2^{\text{domopt}}\right)$. However, the linear structures of a Boolean function will appear in its Walsh spectrum as vector subspaces which contain all the non-zero Walsh coefficients, as was highlighted in the Noekeon attack (see Subsection 5.4.3). Furthermore, Walsh transform pruning can exploit additional properties which are not detected by domopt. For this reason, we think that Walsh transform pruning works better in this case.

6.4 Differential-linear Attack on Reduced-round Serpent

In this section of the chapter, we will cover an application example which we introduced in [BCD+22]. It is a differential-linear attack on 12-round Serpent [BAK98]. Other attack examples were also presented in [BCF+21] and its supplementary material, such as a related-key rectangle attack on GIFT, a meet-in-the-middle attack on PRESENT, a differential attack on RECTANGLE, and the linear attack on Noekeon which was described in Section 5.4.3.

The 12-round differential-linear attacks on 256-bit Serpent, which are compared to previous attacks in Table 6.3, are based on a 12-round differential-linear attack which was proposed by Dunkelman et al. [DIK08]. This attack is shown to be flawed. We first propose a way to correct the error without using decision trees with a time complexity barely smaller than exhaustive search, and then we use the trees to propose an improved version with reduced data and time complexities.

6.4.1 Serpent Specification and Previous Cryptanalysis

Serpent is a block cipher which was introduced by Biham, Anderson and Knudsen [BAK98]. It's designed as a key-alternating block cipher with an internal state of 128 bits. It admits 128, 192 or 256-bit keys. The encryption map consists of a round function which is iterated 32 times. The round function consists of three steps: first the state is XORed with key material, then a layer of of 4-bit Sboxes is applied, and the round ends with a linear transformation. The cipher makes use of 8 different Sboxes (see Table 6.4) which are alternated between different rounds.

The 128-bit internal state X is represented by four 32-bit words denoted X_0, X_1, X_2 and X_3, with

Algorithm 19: The Serpent block cipher

Input: A 128-bit plaintext P, 33 round subkeys $K_0, \ldots, K_{32}$.
Output: A 128-bit ciphertext C.
$\widehat{B}_0 \leftarrow IP(P)$;
for $i \leftarrow 0$ **to** 30 **do**
  $\widehat{B}_{i+1} \leftarrow LT(\widehat{S}_i(X \oplus \widehat{K}_i))$;
end
$\widehat{B}_{32} \leftarrow \widehat{S}_i(X \oplus \widehat{K}_i) \oplus \widehat{K}_{32}$;
return $IP^{-1}(\widehat{B}_{32})$;

$X_j[i]$ being the i-th leftmost bit of word j. The 32 rounds are numbered 0 to 31. We denote by $\widehat{B}_i$ the full 128-bit state at round i. Each round consists of the following four steps:

- **Key mixing:** A 128 bit subkey is XORed to the internal state. The subkey at round i will be denoted by $\widehat{K}_i$.
- **Sbox Layer:** Different Sboxes are used depending on the round, in particular $S_{i \bmod 8}$ is used at round i. The Sbox Layer operation consists of the application of 32 copies of the Sbox to the internal state. For each $i \in \{0, \ldots, 31\}$ we perform the appropriate Sbox transformation to the 4-bit string $(X_0[i], X_1[i], X_2[i], X_3[i])$. The parallel application of the Sbox in round i will be denoted $\widehat{S}_i$.
- **Linear transformation:** The linear operation is denoted by LT and consists of the steps:

$$
\begin{aligned}
&X_0 \leftarrow X_0 \lll 13; \quad X_2 \leftarrow X_2 \lll 3;\\
&X_1 \leftarrow X_1 \oplus X_0 \oplus X_2; \quad X_3 \leftarrow X_3 \oplus X_2 \oplus (X_0 \ll 3);\\
&X_1 \leftarrow X_1 \lll 1; \quad X_3 \leftarrow X_3 \lll 7;\\
&X_0 \leftarrow X_0 \oplus X_1 \oplus X_3; \quad X_2 \leftarrow X_2 \oplus X_3 \oplus (X_1 \ll 7);\\
&X_0 \leftarrow X_0 \lll 5; \quad X_2 \leftarrow X_2 \lll 22;
\end{aligned}
$$

  In round 31 this linear transformation is omitted and an additional whitening subkey $\widehat{K}_{32}$ is XORed to the state instead.

The cipher operates as Algorithm 19, where IP denotes a fixed initial permutation.

The key schedule turns the 256-bit user key K into 33 128-bit round subkeys, which are treated as 132 32-bit words of key material. The user key is first rearranged into a prekey sequence of 8 32-bit words $K = w_{-8}w_{-7}\cdots w_{-1}$. The sequence is extended using the recurrence:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11, \quad 0 \le i \le 131,$$

where $\phi =$ `0x9e3779b9`. We then build the sequence k_i from w_i using the Sboxes:

$$
\begin{aligned}
\{k_0, k_{33}, k_{66}, k_{99}\} &= S_3(w_0, w_{33}, w_{66}, w_{99})\\
\{k_1, k_{34}, k_{67}, k_{100}\} &= S_2(w_1, w_{34}, w_{67}, w_{100})\\
&\cdots\\
\{k_{31}, k_{64}, k_{97}, k_{130}\} &= S_4(w_{31}, w_{64}, w_{97}, w_{130})\\
\{k_{32}, k_{65}, k_{98}, k_{131}\} &= S_3(w_{32}, w_{65}, w_{98}, w_{131})
\end{aligned}
$$

We then distribute the 32-bit words k_j to build the 128-bit subkeys K_i:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

The round key $\widehat{K_i}$ is the result of the application of the fixed initial permutation IP to K_i, hence $\widehat{K_i} = IP(K_i)$.

Serpent has been the target of multiple reduced-round cryptanalysis efforts, including linear [BDK01a, CSQ07b], multiple and multidimensional linear [CSQ07a, CHN08, NWW11], nonlinear [MC13], (amplified) boomerang [KKS00, BDK01a, BDK02b, BDK02a] and differential-linear [BDK03, DIK08, LLL21] attacks. We will now briefly discuss the best of these attacks, which we consider to be the ones which claim to attack 12 rounds of the 256-bit key version of Serpent (see Table 6.3).Unfortunately, some of

these published attacks are flawed or have wrong complexity estimations. In some cases the mistakes have already been described in the literature, but not for all of them. We contacted the authors of all the concerned papers and had them confirm that their original complexity estimates were indeed too optimistic.

The first claimed 12-round attack was proposed by Dunkelman et al. [DIK08], who proposed some differential-linear attacks on 11 and 12-round Serpent. However, we have found that the 12-round attack is incorrect, as was also found independently by Liu et al. [LLL21]. The attack extends the 11-round variant by adding a 112-keybit guess in the first round. However, this guess is not enough, as although the difference in the state after the first round is determined, the input values of the active Sboxes in the second round are unknown, and they are required by the 11-round attack. The attacks we presented in [BCD+22] and reproduced later in this section are amended versions of this attack, as they are based on the same distinguisher.

A family of multidimensional linear attacks on up to 12 rounds was presented by Nguyen et al. [NWW11]. The complexity estimates for these attacks were found to be overly optimistic [MC13], as they rely on an overestimated capacity, erroneous time complexity conversion to full encryptions, and small advantage and success probability. Of the two proposed variants, the complexities of the first were corrected, while the second variant was found to be invalid and no corrected version was described. We have found, however, that the corrected memory complexity for the first attack given in [MC13] was also incorrect. The attack consists of 2^{128} repetitions of the 11-round attack, once for each guess of the last round subkey. We can choose between a memory complexity of $2^{125.813}$ to store the data but with a larger time complexity of $2^{253.813}$, or a large memory complexity of 2^{236} with the same time complexity. The authors of [NWW11] were contacted, but they were unable to provide any further insight. The authors of [LLL21] were also contacted and agreed with our assessment in a personal communication.

Independently of our work, Liu et al. have proposed a new 12-round differential-linear attack [LLL21] using an algebraic approach. In the original paper, the authors claimed a memory of 2^{99}, which ignores the cost of storing the data (as each plaintext-ciphertext pair has to be accessed multiple times). We contacted the authors, who agree with this correction.

6.4.2 Fixing the Attack of Dunkelman et al.

We will first present a corrected version of the flawed 12-round attack from [DIK08] which considers the correct diffusion in the key recovery rounds along with the idea introduced in [BLT20] of generating multiple good differential pairs. We use the same differential-linear distinguisher as the flawed 12-round attack. The attack is built around a "central" distinguisher which starts with a fixed difference at the beginning of round 2, progresses through a three-round truncated differential with probability $p = 2^{-6}$, and then ends with a five-round linear trail with correlation $q = 2^{-21}$ (we remove the last round of the original distinguisher). The expected correlation for the differential-linear distinguisher is thus $pq^2 = 2^{-48}$. However, experiments with the correlation of the transition rounds between the differential and the linear trail suggest that the actual correlation should be at least $2^{-46.75}$. Our aim for the 12-round attack is to effectively extend this distinguisher by two rounds at the top and two rounds at the bottom.

We first try to extend this distinguisher by adding an extra round to the differential at the top and an extra round to the linear trail at the bottom. If we consider an optimal differential transition for each active Sbox in round 1 (with probability 2^{-12}) and an optimal linear approximation for each active Sbox in round 10 (with correlation 2^{-5}) without any key recovery, the data needed for a reasonable probability of success would be more than $2^{2\cdot(48.75+12+2\cdot 5)} = 2^{141.5}$, which surpasses the size of the codebook. This version of the attack is illustrated in figure 6.3.

Our first improvement is inspired by the ideas proposed in [BLT20] in the context of ARX, which we adapt to SPN constructions, and aims to improve the differential probability by ensuring that some parts of the differential transition always hold. We will make sure that some of the differential transitions in round 1 hold for all the data by determining the input values to these Sboxes, instead of only the input differences. In fact, the input difference to these round 1 Sboxes is considered not to be fixed, as we are only interested in the output difference. The end result is that the input difference to the differential is staggered between rounds 1 and 2. This has a cost in additional key recovery bits, but allows us to reduce the data complexity.

In the base attack there are five active Sboxes (11,14,17,18 and 31) in round 1, which then activate 20 Sboxes in round 0. In other words, we have to guess 80 bits of $\widehat{K}_0$ in order to obtain the desired difference at the beginning of round 1. The aim now is to choose some Sboxes in round 1 for which we will also determine the input values, and to do so in a way which optimizes the number of additional keybit guesses. We therefore look at each of the active Sboxes in round 1 and see which additional

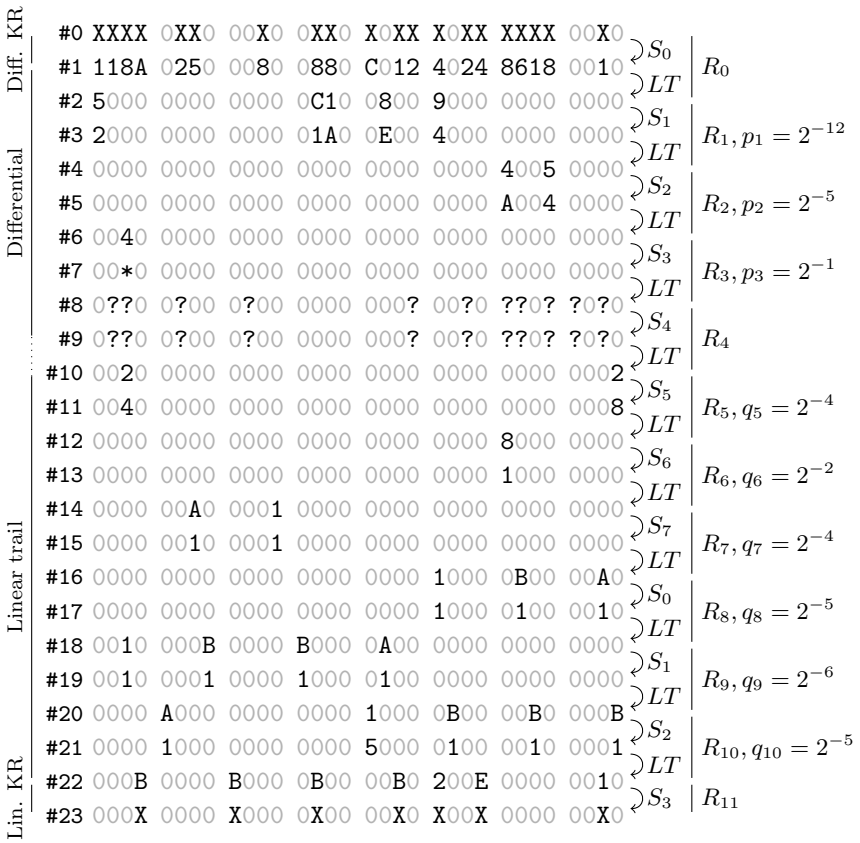


Figure 6.3: The basic differential-linear attack on 12-round Serpent obtained by extending the distinguisher, which requires more data than the whole codebook. The zero nibble differences and masks are written in grey to improve readability. Nibble differences and masks marked by `X` are undetermined. The nibble difference `*` is undetermined, but is zero in the rightmost bit.

Sboxes in round 0 would become active if we decide to guess the input values as well as the differences. These additional active Sboxes in round 0 are given in Table 6.5. It is clear that our efforts should be directed towards Sboxes 11, 17,18 and 31, as they leave the most inactive Sboxes. Sboxes 31 and 17 are particularly interesting as the differential probability is smaller. Giving this treatment to the four Sboxes would make all Sboxes except for 23 active in round 1, so we need to find a compromise.

Given these considerations we decide to determine the input values to Sboxes 31, 18 and 17, and to additionally impose two conditions on the input differences which reject some of the data but make the key recovery less costly. We first impose a 1-bit condition to the difference in state #2 by requiring the difference in bit x_3 in column 17 to be equal to 0. This rejects half of the plaintext pairs, but reduces the number of active bits in the previous round. This condition is compatible with the original nibble difference, which was 1. In addition, in column 31 we only consider differences in which the difference in x_0 is zero. This condition is not compatible with the original nibble difference of 5 and rejects 3/4 of the plaintexts, but makes column 29 in the previous round inactive. In the end, we keep 2^{-3} of the plaintext pairs, which means we can generate up to 2^{124} differential pairs for each key guess from the whole codebook. This new version of the differential is shown in Figure 6.4.

In addition to the 76 active keybits corresponding to determining the difference at the input to round 1 (4 bits are gained with respect to the original 80 because column 29 is no longer inactive), determining the input values to Sboxes 31, 18 and 17 requires guessing all the rest of the round subkey $\widehat{K}_0$ except for Sboxes 23 and 29, which are inactive. This implies a 120 bit key guess in round 0 plus the 12 active bits in round 1 which correspond to Sboxes 31, 18 and 17. In the end, the amount of pairs required by the attack is reduced by a factor of $2^{2\cdot(3+3+2)-3} = 2^{13}$ with respect to the previous version.

The following pair generation procedure is repeated until enough pairs are available:

1. Generate a random plaintext x.
2. Guess 120 bits of $\widehat{K}_0$ and 12 bits of $\widehat{K}_1$. With these we can compute the outputs of S_1 in columns 17, 18 and 31, which will be denoted by $y[17], y[18]$ and $y[31]$, respectively.

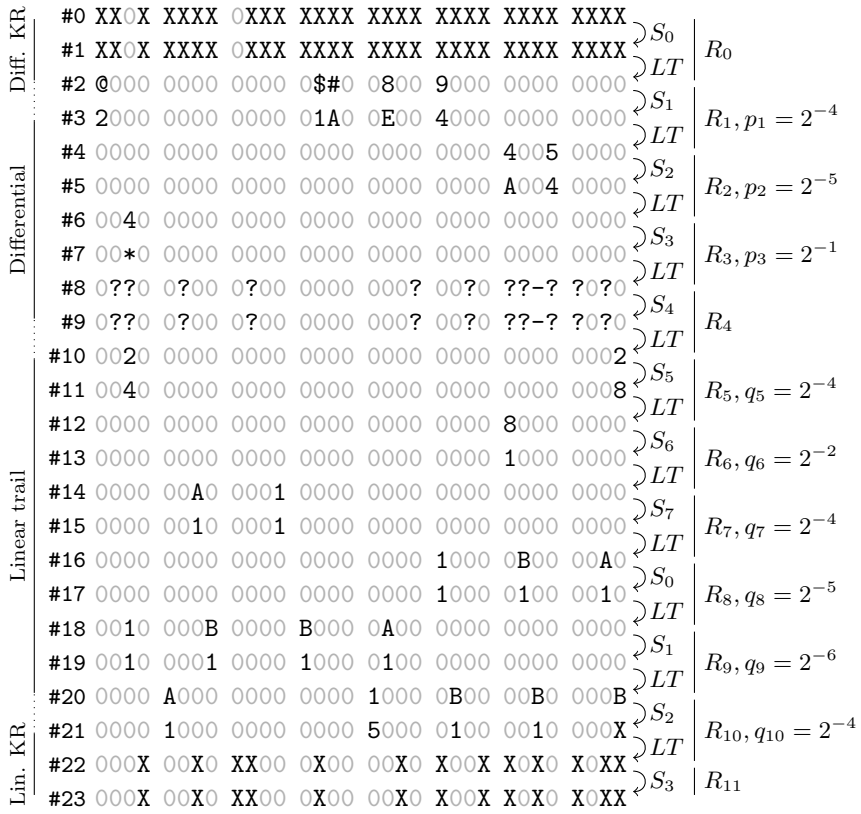


Figure 6.4: The slightly improved differential-linear attack on 12-round Serpent with staggered key-recovery. The nibble difference `$` is undetermined but is one in the leftmost bit x_3 because of the differential properties of S_1. The nibble difference `@` is undetermined but is zero in the rightmost bit x_0. The nibble difference `#` is undetermined but is zero in the leftmost bit x_3.

3. These four nibbles at the output of S_1 are used to compute

$$\Delta[17] = S_1^{-1}(y[17]) + S_1^{-1}(y[17] + \texttt{A}), \tag{6.22}$$
$$\Delta[18] = S_1^{-1}(y[18]) + S_1^{-1}(y[18] + \texttt{1}), \tag{6.23}$$
$$\Delta[31] = S_1^{-1}(y[31]) + S_1^{-1}(y[31] + \texttt{2}), \tag{6.24}$$

   which are the input differences to these Sboxes which will produce the desired output differences. If $\Delta[17]_3 = 1$ or $\Delta[31]_0 = 1$, we reject the plaintext for this key guess.

4. Together with the fixed input differences $\Delta[11] = \texttt{9}$ and $\Delta[14] = \texttt{8}$, we have obtained the appropriate input difference for round 1, which we denote Δ.

5. The associated plaintext is $x' = S_0^{-1}(S_0(x + \widehat{K}_0) + LT^{-1}(\Delta)) + \widehat{K}_0$.

Next, we will have a look at round 10 to try and improve the correlation of the linear trail by staggering the output mask between rounds 9 and 10, similarly to what was done for the input difference. For each of the five active Sboxes (0,5,10,15,27), we determine which Sboxes would remain inactive in round 11 if we determined the full value at the output instead of just the parity bit associated to the linear approximation. The results are shown in Table 6.6. We decide to determine the full output of Sbox 0 and consider the other four active Sboxes in round 10 as part of the differential-linear distinguisher. In this situation, there are 13 active Sboxes in the last round, which means that we need to guess $4 \cdot 13 = 52$ keybits of $\widehat{K}_{12}$, and 4 bits of $\widehat{K}_{11}$.

The key recovery in the linear part can be performed efficiently by using the Walsh transform technique of Chapter 5, at a cost of $2 \cdot 52 \cdot 2^{2 \cdot 52}$ additions/subtractions and $2^{2 \cdot 52}$ products for each key guess in the differential pair generation and each of the 2^4 key guesses in round 10.

In order to properly evaluate the time complexity, we need to compare the cost of the basic operations of the attack against 12-round Serpent encryptions. We first focus on the pair generation and the distillation phase of the FFT algorithm. For each generated plaintext, we perform a two round encryption

Table 6.5: The five Sbox differential transitions in round 1, and the Sboxes which remain inactive in round 0 if we determine the full 4-bit input to each of these Sboxes.

R1 Sbox	Diff. Prob.	Sbox Inactivity																													
		31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
None	-					✓			✓	✓	✓		✓	✓			✓		✓				✓							✓	✓		✓
11	2^{-2}					✓				✓	✓			✓			✓													✓		
14	2^{-2}										✓		✓	✓																		
17	2^{-3}					✓				✓	✓								✓													✓
18	2^{-2}								✓	✓				✓									✓								✓	
31	2^{-3}									✓									✓				✓									✓

Table 6.6: The five Sbox linear approximations in round 10, and the Sboxes which remain inactive in round 11 if we determine the full 4-bit output to each of these Sboxes.

R10 Sbox	Corr.	Sbox Inactivity																													
		31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
None	-	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓		✓	✓	✓	✓	✓	✓		✓
0	2^{-1}	✓	✓	✓		✓	✓		✓			✓	✓	✓		✓	✓	✓	✓		✓		✓	✓			✓		✓		✓		
5	2^{-1}	✓		✓			✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓					✓					✓	✓			✓
10	2^{-1}	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓			✓		✓					✓				✓	✓		✓		
15	2^{-1}	✓	✓	✓		✓	✓	✓	✓			✓		✓					✓				✓	✓		✓	✓		✓	✓	✓		✓
27	2^{-1}	✓					✓	✓			✓	✓		✓			✓	✓	✓		✓		✓	✓		✓	✓	✓	✓	✓			

at the beginning and a one round decryption at the end. However, the partial encryption can be performed for one plaintext x and the result can be reused for all input values of the two inactive Sboxes in round 0. The filtering step requires us to process 2^3 plaintexts to find one which satisfies the desired conditions. The average cost of processing each plaintext pair is thus $(2^{-8} \cdot 2^3 \cdot 2/12 + 1/12) = 2^{-3.50}$ encryptions.

For the comparison of the cost of the arithmetic operations of the Walsh transform to a 12-round encryption, we note that we can consider $128 \cdot 3 \cdot 12 = 4608$ as a lower bound for the number of bit operations in a 12-round Serpent encryption. On the other hand, a 128-bit addition can be performed with 256 bit operations and a 128-bit product with $128 \cdot \log_2(128) \simeq 961$ bit operations. We obtain a worst-case $2^{-4.17}$ factor for the additions and $2^{-2.26}$ for the products.

Given the distinguisher's correlation of $2^{-48.75-4-2\cdot 4} = 2^{-60.75}$, and using the model from [BN16], we obtain that with $2^{123.96}$ pairs, ($2^{127.96}$ data before sieving), an advantage of 15 bits is obtained with probability 0.1. The time complexity of the attack is as follows:

$$\underbrace{2^{120} \cdot 2^{12}}_{\text{Top key guess}} \cdot \Big(2^{-3.5} \cdot 2^{123.96} + \underbrace{2^4 \cdot \left(2^{-4.17} \cdot 104 + 2^{-2.26}\right) \cdot 2^{104}}_{\text{Bottom key guess}}\Big) + 2^{256-15} \simeq 2^{252.46} \tag{6.25}$$

encryptions. We need $2^{127.96}$ memory registers to store the data, and an additional 2^{104} registers for the distillation tables of the FFT. The overall memory complexity is thus around $2^{127.96}$.

6.4.3 Improved Attack using BDDs

This subsection discusses the application of the binary tree technique to the fixed version of the 12-round attack, which before optimisation has a time complexity of $2^{252.46}$ encryptions, which is barely smaller than exhaustive search. The reduction in time complexity is achieved by a careful analysis of the key recovery in the first two rounds, as well as a small improvement to the linear approximation of round 10 using conditional linear cryptanalysis [BP18].

We start by looking at the configuration of required differences and values at the output of S_0 in the first round, which is represented in detail in Figure 6.5. The figure contains both the differences (top half) and the values (bottom half) which are required at the output of the S_0 layer (state #1 in Figure 6.4). The colours of the columns (also indexed with lower case letters from a to f) indicate where we can reduce the amount of keybits we guess below four.

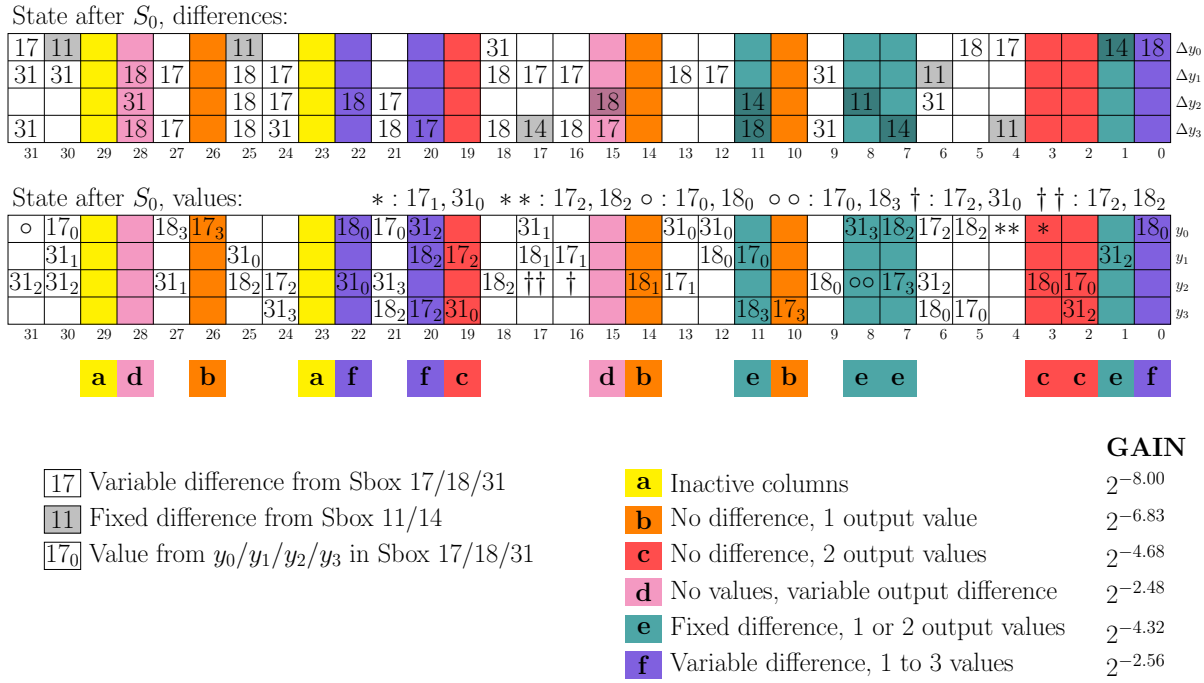


Figure 6.5: Reducing the required keybit guesses in the first round.

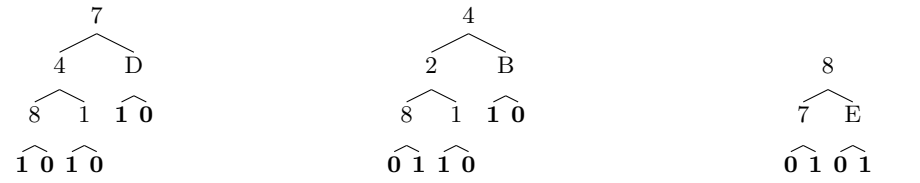
On the top half, the bits represented by cells which have a number written inside will influence the S_1 input difference in state #2 to the Sbox of that number. The bits which influence columns 11 and 14 will always exhibit a difference between both plaintexts in a pair, and are shaded to indicate this. The bits which influence columns 11, 17 and 31 may have a difference which is either zero or 1 depending on the input difference to these Sboxes.

On the bottom half, if an indexed number a_b is written inside a cell, then the bit associated to this cell will influence the value of bit b of the input of Sbox a in state #2. Symbols are used when more than one such bit is influenced. We only need the values associated to columns $31, 18$ and 17 as for the other two Sboxes we only require the input difference.

We consider all the active Sboxes in state #1, immediately after the application of S_0. There are six categories of columns which we consider "of interest", and they depend on the outputs which are required for the attack. For each case, we construct an optimal affine binary decision diagram, and also study the possibility of key absorption into the next round. In each of the categories, we are going to compute the multiplicative time factor that we gain with respect to guessing the full first round subkey $\widehat{K}_0$. For a single Sbox, this gain factor is equal to nblopt/2^4 if no key absorption is considered, and nblopt/2^5 if key absorption is used.

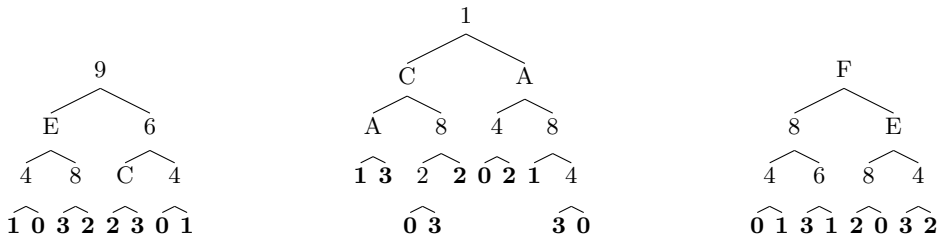
- **Columns of type a (yellow):** There are 2 columns, 29 and 23, which are inactive and need no key guessing at all (the same as in the previous attack). These provide a gain factor of 2^{-8} with respect to a full subkey guess.
- **Columns of type b (orange):** In columns 26, 14 and 10 we require no output differences, but we need to determine a single output bit from each.
  - **Column 26:** We only need y_0, which we can determine with a gain factor of 6/16 using the tree depicted in Figure 6.6a.
  - **Column 14:** We only need y_2, which can be determined with a gain factor of 6/16 using the tree depicted in Figure 6.6b.
  - **Column 10:** We only need y_3, which can be determined with a gain factor of 4/16 using the tree depicted in Figure 6.6c.

  We obtain a total gain factor of $\frac{6}{16} \cdot \frac{6}{16} \cdot \frac{4}{16} \simeq 2^{-4.83}$ for these columns. In addition, we can absorb the last bit guess of some of the columns into the next round. As bit x_3 of column 17 in the input to S_1 is associated to the outputs of both Sboxes 10 and 26 in S_0, we can only do this twice (for the S_1 input bits corresponding to x_3 in column 17 and x_1 in column 18). The overall gain factor with respect to a full subkey guess is thus $2^{-4.83} \cdot 2^{-2} = 2^{-6.83}$.



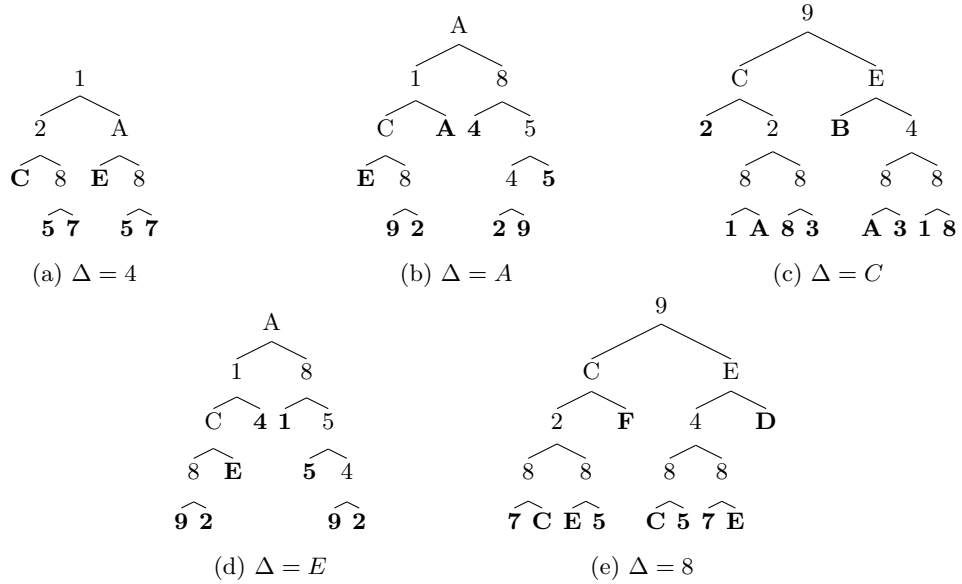
(a) Column 26 (b) Column 14 (c) Column 10

Figure 6.6: Trees used in the columns of type b.



(a) Column 19 (b) Column 3 (c) Column 2

Figure 6.7: Trees used in the columns of type c



(a) $\Delta = 4$ (b) $\Delta = A$ (c) $\Delta = C$

(d) $\Delta = E$ (e) $\Delta = 8$

Figure 6.8: Trees for $F_\Delta = S_0^{-1}(S_0(x) + \Delta) + x$, used in the columns of type d.

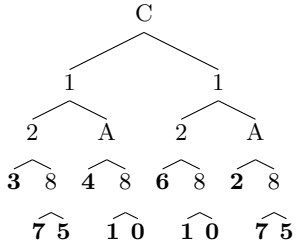


Figure 6.9: The tree used in column 20.

- **Columns of type c (red):** In columns 19, 3 and 2, we require no output difference, but two output bits are needed. We proceed as in the previous case, but considering both bits at the same time.
  - **Column 19:** We need y_1 and y_3. Using the tree in Figure 6.7a, and absorbing one keybit into the next round (x_2 from column 17), we obtain a gain of 2^{-2}.
  - **Column 3:** We need y_0 and y_2. Using the tree in Figure 6.7b, the gain is $2^{-0.68}$.
  - **Column 2:** We need y_2 and y_3. As can be seen in the tree shown in Figure 6.7c, the situation is very similar to that of column 19. We can also absorb one keybit, either x_0 from column 17 or x_2 from column 31. The gain factor is thus also 2^{-2}.

  Combining the gains from these Sboxes we obtain a total factor of $2^{-4.68}$.

- **Columns of type d (pink):** Columns 28 and 15 only require the output difference, but this difference is not fixed. This means that we have to average the costs for each difference.
  - **Column 28:** There are four possible output differences: `0`, `4`, `A` or `E`, which appear with probability 1/4. This can be deduced from the DDT of S_1, the output differences for columns 18 and 31, and the fact that both Δx_1 and Δx_3 come from the difference Δx_2 in column 18. We first guess the parts of the first round subkey which determine the values at the inputs of columns 18 and 31, so that we know which difference we want at the output of column 28. This is only possible because we only need the output differences from these two columns, and no actual bit values.

    We look at the trees for the functions F_Δ shown in Figure 6.8. As an example, when the desired output difference is `4`, the tree has 6 leaves. For the other two nonzero differences, the number of leaves is 8. The overall cost becomes:

    $$\frac{1}{4}6 + \frac{1}{4}8 + \frac{1}{4}2^3 + \frac{1}{4} \cdot 1 \simeq 2^{2.52}$$

    instead of 2^4, which implies a gain factor of $2^{-1.48}$.
  - **Column 15:** We have two possible output differences: `4` and `C` (as the input difference Δx_3 in column 18 is always 1). For the difference `4` the number of leaves is 6, and for `C` it is 10, which gives a total gain factor of $\frac{6/2+10/2}{2^4} = 2^{-1}$.

  Both gain factors multiply to $2^{-2.48}$.

- **Columns of type e (turquoise):** In columns 11, 8, 7 and 1 we have a fixed difference, and we also need to determine some output values. We will once again look at the trees for the functions F_Δ, and incorporate the cost of determining the output bits.
  - **Column 1:** We have a fixed output difference of $\Delta = 1$ and we want to determine bit y_1. If we consider function F_1, we can see that there are 8 possible input differences δ: `3`, `6`, `9`, `C`, `B`, `A`, `E` or `F`. In order to determine whether a pair $(x, x \oplus \delta)$ leads to the desired output difference `1`, we always need to guess three key bits: this is because for any fixed δ the set $\{x : S_0(x) \oplus S_0(x \oplus \delta) = \mathtt{1}\}$ is an affine space (of dimension 1) and therefore can be described by three linear conditions. In all the cases, these three keybits required to determine whether a pair is a good pair are also enough to determine y_1. Column 1 therefore has a gain factor of 2^{-1}.
  - **Column 7:** We proceed in the same way. In some cases, the input differences define an affine space of dimension 2 instead of 3, which means that the gain factor is slightly better at $2^{-1.13}$.
  - **Column 8:** In this case the four input bits are needed, but the key bit x_3 of column 31 can be absorbed, and the gain factor is 2^{-1}.
  - **Column 11:** The four input bits need to be guessed for each input difference, but in half of the cases two of the key guesses will independently determine the two needed output values. In particular, we can use this to absorb y_3, which corresponds to bit x_3 of column 18. The bit y_1, corresponding to bit x_0 of column 17, can be absorbed in the half of cases in which it was not used for the absorption in column 2. The overall gain factor for column 11 is $2^{-1.19}$.

  In total, these columns generate a gain factor of $2^{-4.32}$.

- **Columns of type f (purple):** In columns 22, 20 and 0, we require an optional output difference of a single bit as well as some output values. Depending on the value of this variable difference, we may encounter one of the cases which are already considered.

Table 6.7: The conditional linear approximation of S_2.

	$(y_2, y_0) \neq (1,1)$												$y_2 = y_0 = 1$			
$\boldsymbol{x}$	0	1	3	4	5	6	9	A	B	C	D	F	2	7	8	E
$\boldsymbol{S_2(x)}$	8	6	9	3	C	A	1	E	4	0	B	2	7	F	D	5
$\boldsymbol{\langle B, x\rangle \oplus \langle 1, S_2(x)\rangle}$	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

- **Column 22:** We have two possible output differences: `0` and `4`, and we need y_0 and y_2. When the difference is `0`, we can just use the tree from Figure 6.7b, which gives a factor of 10/16, while when the difference is `4`, the case is similar to that of column 8 (type e), and we obtain a gain factor of 1/2 by absorbing the bit x_0 of column 31. The average gain is $1/2 \cdot \frac{10}{16} + 1/2 \cdot 2^{-1} = 2^{-0.83}$.
- **Column 20:** We have two possible output differences: `0` and `8`, and we need y_0, y_1 and y_3. When the difference is `0`, an optimal tree for these three bits is given in Figure 6.9. From this we obtain a gain factor of $1/2 \cdot 2^{-0.42} + 1/2 = 2^{-0.19}$.
- **Column 0:** The possible output differences are `0` and `1`, and we wish to determine y_0. The situation for the difference `0` is the same as that of column 26 (type b), and we can even absorb bit x_0 of column 18. When the difference is 1, we recover a similar case as for columns of type e, having in this case a gain factor of 1/2. The gain factor of this column is therefore $1/2 \cdot \frac{6}{16} \cdot 1/2 + 1/2 \cdot 1/2 = 2^{-1.54}$.

This gives a total gain factor for these columns of $2^{-2.56}$.

Since the bottleneck of the previous attack was the exhaustive search step, we also need to somehow improve the correlation of the differential-linear distinguisher so that we can increase the advantage. We can improve the last round of the linear approximation using conditional linear cryptanalysis [BP18], which was briefly discussed in 2.3.3.

We consider the Sbox S_2, which is used in round 10, and the linear approximation $\langle \mathtt{B}, x\rangle \oplus \langle \mathtt{1}, S_2(x)\rangle$, which is the one which appears in column 10. As we can see in Table 6.7, if we consider the whole domain, the correlation is $\frac{4-12}{16} = -\frac{1}{2}$. However, under the condition $(y_2, y_0) \neq (1,1)$ the correlation increases to $\frac{2-10}{12} = -\frac{2}{3}$.

We apply this conditional linear approximation to column 10 in round 10. This would improve the overall data complexity by a factor of $\left(\frac{2/3}{1/2}\right)^4 \simeq 2^{1.66}$. As we have to discard the 1/4 ciphertext pairs which do not verify the condition, we only keep a proportion of $(3/4)^2 = 2^{-0.83}$. With this we can reduce the data complexity by a factor of $2^{0.83-1.66}$, resulting in a total $2^{123.13} \cdot 2^4 = 2^{127.13}$. However, since our objective is to improve the time complexity, we instead keep a similar data complexity of $2^{123.92} \cdot 2^4 = 2^{127.92}$ but we increase the advantage to 23 at the same success probability of 0.1.

Regarding the key recovery in the final rounds, the only change with respect to the previous attack is that we require the output bit y_2 of column 10 in round 10. This additional bit means that column 6 in the next round becomes active, thus increasing the number of active keybits in the last round to 56.

The data and memory complexities of this version of the attack are $2^{127.92}$. If we consider all the gain factors we have accumulated, the time complexity of the key recovery step is

$$2^{140-8-6.83-4.68-2.48-4.32-2.56} \cdot \left(2^{-3.5} \cdot 2^{123.92} + 2^4 \cdot \left(2^{-4.17} \cdot 112 + 2^{-2.26}\right) \cdot 2^{112}\right) \simeq 2^{231.91} \tag{6.26}$$

equivalent encryptions, while the exhaustive search has cost 2^{256-23}. The total time complexity is thus around $2^{233.55}$ 12-round encryptions.

6.4.4 Other Complexity Trade-offs

We can reduce the data and memory complexities of the attack by relaxing one of the bit conditions imposed on the difference in state #2 which filtered the data, at the cost of increasing the time complexity. For example, by removing the condition on the difference in column 31, the data complexity is reduced by a factor 2^2. This changes the output differences of columns 29, 26 and 13 of round 0.

- **Column 29:** It was previously a column of type a with a gain factor of 2^{-4}, but it now becomes a column of type e. For the 3/4 of the cases where the previous bit condition in column 31 no

longer holds, we must look at F_8, whose optimal tree is shown in Figure 6.8e. The new cost for this column is $1/4 \cdot 1 + 3/4 \cdot 10 = 2^{2.954}$, which results in a more modest gain factor of $2^{2.954-4} = 2^{-1.04}$.

- **Column 26:** It was previously a column of type b with a gain factor of $6/16 = 2^{-1.415}$, which will now only apply to the 1/4 of the data for which the bit condition in column 31 holds. For the other 3/4, the guessing cost is 12/16, giving us an average cost of $1/4 \cdot 6 + 3/4 \cdot 12 = 2^{3.39}$, and a new gain factor of $2^{-0.6}$.
- **Column 13:** Since we didn't use any optimisations on this column, the change doesn't have an influence in the time complexity.

By considering the new gain factors and readjusting the data complexity and advantage, as well as the cost of generating each pair, we obtain a time complexity of

$$2^{140-4-1.04-5.40-0.6-4.68-2.48-4.32-2.56} \cdot \left(2^{-3.56} \cdot 2^{123.74} + 2^4 \cdot \left(2^{-4.17} \cdot 112 + 2^{-2.26}\right) \cdot 2^{112}\right) \qquad (6.27)$$

equivalent encryptions for the key recovery and 2^{256-21} encryptions for the exhaustive search, which result in an overall time complexity of $2^{236.31}$. This time the data and memory complexities are $2^{123.74} \cdot 2^2 = 2^{125.74}$.

We propose an additional trade-off which gives the best data complexity. We determine the full input values to columns 31, 18, 17 and 11 (instead of just 11, 17 and 18), and impose no conditions on their input differences. We can proceed as in the previous attack, and we obtain:

- One column of type a: 23, with a gain factor of 2^{-4}.
- Three columns of type c: 19, 3 and 2, with gain factors $2^{-1.41}$, $2^{-0.415}$ and $2^{-0.415}$.
- Three columns of type e: 8, 7 and 0, with gain factors $2^{-1.678}$, $2^{1.41}$ and $2^{-1.54}$.
- Four columns of type f: 29, 26, 22, 12, with gain factors $2^{-1.607}$, $2^{-1.192}$, $2^{-1.35}$, 2^{-1}.

This gives a total gain factor of $2^{-16.023}$ with respect to a full subkey guess. We will not consider the conditional linear property, as the Walsh transforms substitutes the exhaustive search as the bottleneck of the attack. In the end, for a data and memory complexity of $2^{118.40}$, we can achieve an advantage of 16 bits with probability 0.1, which results in a time complexity of

$$2^{128+16-16.023} \cdot \left(2^{-3.57} \cdot 2^{118.40} + 2^4 \cdot \left(2^{-4.17} \cdot 2104 + 2^{-2.26}\right) \cdot 2^{104}\right) + 2^{256-16} = 2^{242.93}$$

equivalent 12-round Serpent encryptions.

6.5 Conclusion and Open Problems

The tree-based techniques which exploit the properties of Sboxes in order to improve the time complexity of key recovery attacks have been successfully applied to a variety of attacks of different families on multiple ciphers, as shown in Table 6.8, which summarises all the applications we described in [BCF+21, BCD+22]. All of these attacks were obtained by optimising the key recovery step of existing attacks (in other words, we recycled distinguishers from the literature). We speculate that it might be possible that these optimisations might also allow more effective distinguishers to be used than would be otherwise possible, which would in turn result in an improvement to the data complexity as well.

At the time of writing, the main limitation of this approach is the cost of finding optimal trees for large Sboxes of more than 8 bits. This doesn't just exclude some constructions, but it also makes the process of application to a given attack quite technical, as the cryptanalyst needs to try a lot of different approaches until one of them works. Ideally, we would like to be able to process large parts of the cipher construction automatically. This would require algorithms for finding optimal trees which are more efficient than the one we have already described.

Finally, we would like to propose some open problems regarding the theory of the description of Sboxes as affine BDDs. In particular, we think that it would be interesting to find lower bounds for the complexity metrics for the different families of functions which appear in cryptology - for instance, balanced functions. Another problem is that of exploring the relationship between avgdepth and nbleaves.

Table 6.8: Overview of the different attacks which were described in [BCF+21, BCD+22]. [JZZD20] was the best attack on GIFT-64 at the time of writing [BCF+21]. An attack on 26-round GIFT was presented in [SWW21].

					Base attack			**New attack**			
Cipher	n	κ	r	**Type**	**Data**	**Time**	**Source**	**Data**	**Time**	**Source**	**Best?**
Noekeon	128	128	12	Linear	2^{124}	2^{124}	[DPVAR00]	2^{119}	$2^{122.14}$	[BCF+21]	Yes
GIFT	64	128	25	Related-key rectangle	$2^{63.78}$	$2^{120.92}$	[JZZD20]	$2^{62.73}$	$2^{99.18}$	[BCF+21]	No
RECTANGLE	64	80	18	Differential	2^{64}	$2^{78.88}$	[ZBL+14]	2^{64}	2^{64}	[BCF+21]	Yes
PRESENT	64	80	8	Sieve-in-the-middle	2^{6}	$2^{73.42}$	[CNV13]	2^{6}	$2^{72.91}$	[BCF+21]	No
Serpent	128	256	12	Differential-linear	-	-	[DIK08]	$2^{127.92}$	$2^{233.55}$	[BCD+22]	Time
Serpent	128	256	12	”	-	-	[DIK08]	$2^{127.92}$	$2^{236.91}$	[BCD+22]	No
Serpent	128	256	12	”	-	-	[DIK08]	$2^{118.40}$	$2^{242.93}$	[BCD+22]	Data

Chapter 7

Finding Linear Approximations of Gimli using MILP Methods

This chapter contains a more thorough explanation of the techniques used to find linear approximations of reduced-round variants of the Gimli permutation in the co-authored paper with Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher and Ferdinand Sibleyras [FLN+20]. In [FLN+21], we described a linear approximation of the full 24-round permutation. These linear approximations were found by solving Mixed-integer Linear Programming (MILP) optimisation problems using the SCIP solver [VG18].

Contents

A short overview of bit-oriented MILP models of linear trails and differential characteristics for SPNs is provided in Section 7.1. It covers the general way these models are constructed and used, and very briefly discusses the simplest ways in which linear transformations and Sboxes can be incorporated into them. Section 7.2 focuses on the Gimli permutation, containing its specification and an explanation of how the MILP model describing the linear trails of the SP-box was constructed. Finally, it provides a step-by-step description of the process by which the 24-round linear trail of Gimli was constructed.

7.1 MILP-based Differential/Linear Trail Search

The term Mixed-integer Linear Programming or MILP refers to optimisation problems in which all constraints as well as the objective function are linear (hence the "linear programming" part), with the added twist that some or all of the decision variables are only allowed to take integer values instead of real values (hence the "mixed-integer" part).

Definition 7.1. *A* Mixed-integer Linear Programming *or MILP problem is an optimisation problem which can be rewritten in the following* canonical form *by choosing a matrix $A \in GL(\mathbb{R}^m, \mathbb{R}^n)$, a vector $c \in \mathbb{R}^n$, and a subset $\mathcal{S} \subseteq \{1, \ldots, n\}$ appropriately:*

$$\textit{Find } x \in \mathbb{R}^n \textit{ verifying } \begin{cases} Ax \leq 0 \\ x \geq 0 \\ x_i \in \mathbb{Z} \textit{ if } i \in \mathcal{S} \end{cases} \textit{ which maximises } \langle c, x \rangle. \tag{7.1}$$

The terminology is inherited from linear programming. Each coordinate of x is thus called a decision variable*. Each row of the matrix A provides a linear inequality on x, which we call* constraint*. The subset $\mathcal{S}$ indicates the* integer variables*. The set of x which verify all the constraints is called the* feasible region*. Finally, the map $\langle c, \cdot \rangle$ is called the* objective function*.*

Because of its interest to researchers and businesses alike, a variety of solvers for MILP problems exist, such as SCIP [VG18]. For the purposes of this text, these solvers will be considered as black boxes. There are many parameters which can be altered by the user and which can greatly impact performance, but they are beyond our scope. However, we still provide a very broad overview of how these solvers work which can be useful to the cryptanalyst.

In general, they first apply a *presolver* which transforms the problem into an equivalent one which is more friendly to manipulate later. For instance, it checks for variables and constraints which are clearly redundant and removes them from the problem. In solvers which accept problems with equality constraints as an input, these may be turned into an equivalent set of inequalities. After the equivalent problem has been formulated, the main solving phase begins. The solver first finds a solution, and then explores the feasible region, progressively finding solutions which gradually improve the value of the objective function. The value of the objective function for the best current solution is the *primal bound*. At the same time, the solver also provides the *dual bound*, which bounds the values the objective function can take from the other side. As the solver continues running, the gap between the primal and the dual bound decreases, until it (hopefully) reaches zero, at which point the current solution is optimal.

The idea behind using MILP to find differentials or linear trails is as follows: we first construct an MILP problem whose feasible region represents the space of linear trails or differential characteristics of the target cipher. We use a binary variable (that is, an integer variable which is restricted to take the value either 0 or 1) to represent the activity of each individual bit in each internal state. With the help of some well-chosen constraints and possibly some dummy variables, we can then represent the possible transitions for each component of the cipher, which removes all trails of correlation zero or all differentials of probability zero from the search space. In other words, each solution to the problem will be a linear trail or a differential characteristic.

At the nonlinear components of the construction, we add some auxiliary variables which indicate when this element is active and makes a contribution to the differential probability or correlation. For example, for each Sbox we may add a variable which indicates whether this Sbox is active (1) or not (0). By adding up all these indicator variables, we construct an objective function which is equal to the number of active Sboxes. Finding a solution which minimises this objective function is equivalent to finding a trail with minimal number of active Sboxes. By associating more than one variable with each Sbox, we can indicate, in addition to whether the Sbox is active or not, the differential probability or correlation of its current transition. By minimising a properly weighted objective function, we can then obtain trails which have maximum correlation or differential probability. We must also add a constraint which removes the trivial zero trail from the feasible region, either by adding a lower bound of 1 on the objective function, or by forcing the sum of the bit activity indicators in the initial state to be greater than zero.

This idea was first introduced as a word-oriented model by Mouha et al. [MWGP11]. This model could be used to find lower bounds for the number of active Sboxes in trails of word-oriented block ciphers using optimisation problems with a relatively small number of variables and constraints. However, this word-oriented approach is not constructive, in the sense that it cannot be used to build explicit linear trails or differential characteristics. It is also limited to specific cipher constructions. For these reasons, bit-oriented models were introduced by Sun et al. [SHW+14b, SHW+14a]. Although they involve a significantly larger number of variables and constraints and may thus take more time to solve, they provide a more detailed description of the cipher and can thus be used to obtain full linear trails or differential characteristics as their solutions, as well as proving their optimality.

In the rest of this section, we will describe some simple steps which can be followed to build MILP models of SPNs and similar constructions. Given the scenario described above in which the activity of each bit of each intermediate state is indicated by one binary variable, we just need to describe how we can obtain the constraints which model the behaviour of linear layers, as well as showing how to handle Sbox layers.

7.1.1 MILP Models of Linear Layers

Let us consider a transformation of the state consisting of the multiplication by a fixed matrix $L \in GL(\mathbb{F}_2^n, \mathbb{F}_2^n)$, that is, $y = Lx$. We know that, for differential cryptanalysis, a difference Δ propagates to a difference Δ^* if and only if $\Delta^* = L\Delta$. Similarly, for linear cryptanalysis, we know that the correlation

between an input mask α and an output mask β is non-zero if and only if $\alpha = L^T\beta$, and in that case the correlation is 1. The linear transformation L can also be given as a circuit consisting of branching points and XOR operations. In that case, linear masks and differences propagate according to the rules illustrated in Figure 2.1.

7.1.1.1 Circuit Approach

We can create a model for the linear transformation by giving each wire in the circuit its own activity decision variable and modelling each XOR operation and branching point individually. This is quite effective in the case of lightweight linear transformations, which can be computed with a circuit containing relatively few operations. However, for more complex linear layers, the number of auxiliary variables grows very quickly and may thus make the resulting MILP problem increasingly difficult and costly to solve.

We first look at branching points for differential characteristics, or alternatively, XOR operations for linear trails. Since the activity of the three branches must be the same in order for the differential probability/correlation to be different from zero, it is sufficient to use the same variable for all three branches. In practice, this can be done either by carrying out this substitution directly when designing the model, or by using three different variables and adding two equality constraints which guarantee that they always have the same value. In the latter case, it is left to the presolver to take care of the substitution and turn the three separate variables into a single one.

In XOR operations for differential characteristics and branching points for linear trails, we must ensure that the XOR of the activity of the three branches is equal to zero. Let us assume that the activity variables for the three branches are x, y and z. Then we can represent $x \oplus y \oplus z = 0$ using five constraints and a dummy binary variable d:

$$x \oplus y \oplus z = 0 \quad \Longleftrightarrow \quad \begin{cases} d \geq x,\ d \geq y,\ d \geq z \\ x + y + z \geq 2d \\ x + y + z \leq 2 \end{cases}. \tag{7.2}$$

The first three constraints ensure that if one of the three branches is active, then the dummy variable is 1. In that case, the fourth constraint ensures that another branch is also active. Finally, the fifth constraint ensures that it is not possible that all three branches are active at the same time. Alternatively, the following model without dummy variables can be used:

$$x \oplus y \oplus z = 0 \quad \Longleftrightarrow \quad \begin{cases} -x + y + z \geq 0 \\ +x - y + z \geq 0 \\ +x + y - z \geq 0 \\ +x + y + z \leq 2 \end{cases}. \tag{7.3}$$

7.1.1.2 Matrix Approach

For linear layers which have a more complex circuit description, it might be more worthwhile to construct the MILP model directly from the linear transformation matrix L for differential characteristics and L^T for linear trails. The relationship $y = Mx$, where $M \in GL(\mathbb{F}_2^n, \mathbb{F}_2^m)$, consists of m equalities over n variables of the form $y_i = \sum_{j=0}^{n} m_{ij}x_j$. Each one of these equalities is the XOR of $\mathrm{wt}(M_{i.}) + 1$ variables. We can model a linear equation over $\mathbb{F}_2$ of the form $x_1 \oplus \ldots \oplus x_l = 0$ as a set of 2^{l-1} real linear inequalities as follows:

$$x_1 \oplus \ldots \oplus x_l = 0 \quad \Longleftrightarrow \quad \left\{ \sum_{i=1}^{l} (-1)^{u_i} x_i \geq 1 - \mathrm{wt}(u),\ u \in \mathbb{F}_2^l,\ \mathrm{wt}(u) \bmod 2 = 1 \right\}. \tag{7.4}$$

Using this description, we can model each row of the matrix M separately and obtain a model of the linear transformation without any dummy variables. However, this technique requires a large amount of constraints when the matrix M is not very sparse.

Boura and Coggia [BC20] showed that the previous model for the XOR operation has the minimal number of inequalities among models without any additional dummy variables. They also showed that the number of inequalities used to model the whole matrix can be reduced using the following method. By considering the matrix $M' = (M|I_m)$, the equality $y = Mx$ is rewritten as $M'(x|y) = 0$. The number of inequalities required in an MILP model of the transformation is $\sum_{i=1}^{m} 2^{\mathrm{wt}(M'_{i.})}$. However, for any invertible matrix P, the equation $PM'(x|y)$ is also a valid description of the linear transformation. By finding a matrix P for which the Hamming weights $\mathrm{wt}(P_{i.}M')$ are low, models with much fewer constraints can be obtained. The authors also provide an algorithm which finds optimal matrices P.

7.1.2 MILP Models of Sbox Transition Tables

When it comes to modelling Sbox layers, we will consider each application of the Sbox separately. In addition to removing any trails which exhibit transitions with correlation zero from the feasible region of the MILP problem, we also want to construct an objective function which accurately represents either the probability of the differential characteristic or the correlation of the linear trail.

7.1.2.1 The Objective Function

We will start by showing how we can define appropriate auxiliary variables and the objective function. The explanation will be given for the case of the LAT in linear cryptanalysis, but the DDT for differential cryptanalysis can be handled in exactly the same way. Let $S : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be an n-bit to m-bit Sbox. We start by separating the linear approximation table of the Sbox into as many different binary matrices as there are different absolute values for its non-zero entries other than $\mathrm{LAT}_S(0,0)$. For example, if the LAT of a 4-bit Sbox has entries with values 0, ± 2 and ± 4, we can build its 2-LAT and its 4-LAT. The k-LAT is thus defined as

$$k\text{-LAT}_S(\alpha,\beta) = 1 \quad \Longleftrightarrow \quad \mathrm{LAT}_S(\alpha,\beta) = \pm k. \tag{7.5}$$

Let us assume that S has d different k-LATs and that for each one we have found a set of l_k inequalities which represent the k-LAT of the Sbox, that is, some coefficients $\{a_{ij}^k\}_{1\leq i\leq l_k, 1\leq j\leq n}$, $\{b_{ij}^k\}_{1\leq i\leq l_k, 1\leq j\leq m}$ and $\{c_i^k\}_{1\leq i\leq l_k}$ for which

$$k\text{-LAT}_S(x_n,\ldots,x_1,y_m,\ldots,y_1) = 1 \quad \Longleftrightarrow \quad \left\{ \sum_{j=1}^{n} a_{ij}^k x_j + \sum_{j=1}^{m} b_{ij}^k y_j \geq c_i^k,\ 1\leq i\leq l_k \right\}. \tag{7.6}$$

We will cover how these sets of inequalities can be constructed in the coming pages.

Abdelkhalek et al. [AST+17] proposed a way to incorporate all the separate k-LATs at the same time with each having a contribution to the objective function proportional to k. It uses so-called *conditional constraints*. Let M be a large constant which verifies $M > 2(n+m)\max(|a_{ij}^k|, |b_{ij}^k|)$. Then we can add d indicator variables S_k and the following constraints to the model:

$$\sum_k S_k \geq x_1, \ldots, \sum_k S_k \geq x_n,\ \sum_k S_k \geq y_1, \ldots, \sum_k S_k \geq y_m,\ \sum_k S_k \leq 1. \tag{7.7}$$

These constraints make sure that once one input or output bit of the Sbox is active, then exactly one of the S_k indicator variables must be 1. The variable S_k being active means that the Sbox is active and has correlation $\pm k2^{-n}$. For each k, we will add the following conditional constraints:

$$\left\{ \sum_{j=1}^{n} a_{ij}^k x_j + \sum_{j=0}^{m} b_{ij}^k y_j + M(1-S_k) \geq c_i^k,\ 1\leq i\leq l_k \right\}. \tag{7.8}$$

When $S_k = 0$, the M on the left-hand side dominates all the inequalities and therefore they always hold independently of the values of the x_j and y_j. When $S_k = 1$, the M gets cancelled out and the associated constraint on the x_j and y_j becomes effective.

In order to find linear trails with optimal correlation, it suffices to minimise the following objective function, which is the sum of all the S_k indicator variables for all the Sboxes in the cipher, using weights proportional to the exponent of the correlation. This means this objective function is minus the exponent of the correlation of the linear trail.

$$\text{Minimise} \sum_S \sum_k -\log_2\left(k2^{-n}\right) \cdot S_k. \tag{7.9}$$

Finally, we add the following constraint, which ensures that the trivial zero linear trail is excluded from the search space:

$$\sum_S \sum_k S_k \geq 1. \tag{7.10}$$

7.1.2.2 MILP models of transition tables using Karnaugh maps

The only missing piece in the procedure to construct MILP models of linear and differential trails of SPN constructions is that of finding good descriptions of a k-LAT or k-DDT of an Sbox as a set of real

inequalities. There are several different methods which can be used to generate large sets of inequalities which model the Sbox transitions, as well as methods which can find optimal selections of inequalities, so that the size of these sets can be reduced.

The two most commonly-used methods were also introduced in [AST+17]: one of them is based on algorithms which can compute a description of the convex hull of a set of points, and the other is based on logical condition modelling. In [BC20], new methods as well as improvements were proposed. Since we just intend to provide a brief introduction, we will only discuss the logical condition modelling approach using the Quine-McCluskey algorithm [MJ56], which can be used without much difficulty. Sets of inequalities for Sboxes of up to 4 bits can even be obtained by hand, and in many cases the performance (that is, the running time for the solver) is comparable to that obtained when using other more elaborate methods.

Let us consider the k-LAT of an Sbox $S : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$. We want to obtain a small set of inequalities on the variables $x_n, \ldots, x_1$ and $y_m, \ldots, y_1$ so that each transition for which $k\text{-LAT}_S(x_n, \ldots, x_1, y_m, \ldots, y_1) = 0$ is rejected by at least one inequality. We can simplify the notations by reducing the problem to a more general one. We consider a Boolean function $f : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$, in our case we have $l = n + m$ and $f(\alpha|\beta) = k\text{-LAT}_S(\alpha, \beta)$. Our objective is to obtain inequalities which remove all the points $(x_l, \ldots, x_1)$ at which f is zero without removing any points at which f is 1.

Given disjoint $\mathcal{S}_0, \mathcal{S}_1 \subseteq \{1, \ldots, l\}$, the inequality

$$\sum_{i \in \mathcal{S}_0} x_i - \sum_{i \in \mathcal{S}_1} x_i \geq 1 - |\mathcal{S}_1|, \tag{7.11}$$

removes exactly the $2^{l-|\mathcal{S}_0|-|\mathcal{S}_1|}$ points $x \in \mathbb{F}_2^l$ which verify $x_i = 0$ for all $i \in \mathcal{S}_0$ and $x_i = 1$ for all $i \in \mathcal{S}_1$. Indeed, we can see that if x verifies this condition, then the left term evaluates to $-|\mathcal{S}_1|$, and the inequality doesn't hold. If we have some $i \in \mathcal{S}_0$ for which $x_i = 1$, then its contribution to the sum increases from 0 to 1 (with respect to $x_i = 0$). If for some $i \in \mathcal{S}_1$ we have $x_i = 0$, then its contribution to the sum increases from -1 to 0.

By using this kind of inequality, we can generate sets which remove all the desired inputs, for example by using one inequality to remove each impossible transition (that is, using $\mathcal{S}_0 \cup \mathcal{S}_1 = \{1, \ldots, l\}$ for all inequalities). However, this approach is quite inefficient with regard to the number of inequalities, and results in MILP models which take a long time to solve. For this reason, ways to generate inequalities which remove as many impossible transitions as possible were introduced in [AST+17].

The problem has a lot of similarities to the problem of Boolean function minimisation. Indeed, expressing a Boolean function g as a sum of monomials of the form

$$\bigvee_{(\mathcal{S}_0, \mathcal{S}_1) \in \mathcal{M}} \left(\bigwedge_{i \in \mathcal{S}_0} \overline{x_i} \bigwedge_{i \in \mathcal{S}_1} x_i \right) \tag{7.12}$$

requires that:

- For each $x \in \mathbb{F}_2^n$ with $g(x) = 1$, there is at least one monomial $(\mathcal{S}_0, \mathcal{S}_1) \in \mathcal{M}$ which evaluates to 1 at x, which means that $x_i = 0$ if $i \in \mathcal{S}_0$, and $x_i = 1$ if $i \in \mathcal{S}_1$.

- For each $x \in \mathbb{F}_2^n$ so that $g(x) = 0$, there exist no monomials in the expression which evaluate to 1 at x.

This means we can find small sets of inequalities which reject all the impossible transitions of the k-LAT or k-DDT of an Sbox by minimising the Boolean function $g = \overline{f}$.

Boolean function minimisation is a problem for which several algorithms have been described, with the most widely-known probably being the Quine-McCluskey algorithm [MJ56], which can be used quite effectively for Sboxes of small size. For larger (more than 8 bits) Sboxes, we can use heuristic Boolean function minimisation algorithms such as Espresso.

For Sboxes of up to 4 bits (which means handling 8 variables in total), we can even run the Quine-McCluskey algorithm on the paper-and-pencil architecture using Karnaugh mappings [Kar53], an example of which can be found in Figure 7.3. Broadly speaking, we start by finding valid inequalities of this kind for which the set $\mathcal{S}_0 \cup \mathcal{S}_1$ is smallest, which means we can cover a lot of zeros in the k-LAT with just a few inequalities. Once this is done, we can add increasingly more precise inequalities with larger $\mathcal{S}_0 \cup \mathcal{S}_1$ in order to cover the zeroes which were not covered before, until we have covered all the impossible transitions.

Algorithm 20: The Gimli permutation

Input: Initial state $S = (A, B, C, D)$
Output: Gimli(S)
for $r \leftarrow 24, \ldots, 1$ **do**
  $(A, B, C, D) \leftarrow (\mathrm{SP}(A), \mathrm{SP}(B), \mathbf{SP}(C), \mathrm{SP}(D))$; // SP-Box layer
  if $r \bmod 4 = 0$ **then**
    $\textsc{Swap}(A_x, B_x)$; $\textsc{Swap}(C_x, D_x)$; // Small swap
    $A_x \leftarrow A_x \oplus \texttt{9E377900} \oplus i$; // Round constant
  else if $r \bmod 2 = 0$ **then**
    $\textsc{Swap}(A_x, C_x)$; $\textsc{Swap}(B_x, D_x)$; // Big swap
  end
end
return S;

7.2 Building a Linear Distinguisher of Gimli

This section consists of an extended explanation of the analysis of the linear properties of the Gimli permutation [BKL+17] and its components which was initially published in [FLN+20], and won Best Paper Award at Asiacrypt 2020; and the extended version [FLN+21], as this was my main contribution to these papers. In particular, we will describe how a linear approximation of the full 24-round permutation with an estimated square correlation of $2^{-367.6}$ was found. This was achieved using a bit-oriented MILP model of Gimli constructed using the techniques covered earlier in this chapter, and which was solved using the SCIP solver [VG18].

Gimli [BKL+17] is a 384-bit cryptographic pseudorandom permutation intended towards offering high performance in a wide variety of systems. It is also the basis of the block cipher Gimli-Cipher and the hash function Gimli-Hash, which were presented [BKL+19] as a candidate standard to the NIST Lightweight Cryptography project. It was selected in both the first and the second round, but was not selected as a finalist.

There are several papers which discuss cryptanalysis of the Gimli permutation and its reduced-round variants. The construction's weak diffusion properties were already highlighted in [Ham17], where a meet-in-the-middle type attack on 22.5 rounds which doesn't use any properties of the SP-box was described. Zero-sum distinguishers on up to 14 rounds were presented in [CWZ+19]. We introduced the first full-round distinguisher in [FLN+20], which is what we called an *internal symmetry* distinguisher and has a time complexity of 2^{64} encryptions. A distinguisher on 23 rounds with a practical time complexity of 2^{32} was also given, as well as a linear distinguisher on 16 rounds and a differential-linear distinguisher on 17. In the journal version of this paper [FLN+21], a 24-round linear approximation was provided, which is the one that will be covered in this section. A similar internal symmetry-based full-round distinguisher with time complexity 2^{52} has been described in [LIM21], as well as an 18-round hybrid zero-internal-differential distinguisher.

7.2.1 Description of the Gimli permutation

The Gimli permutation operates on a 384-bit state, which will be denoted by S. The state is subdivided into 4 *columns* of 96 bits, which are denoted A, B, C, D, where A is the leftmost column, and D is the rightmost. Each column is divided into three 32-bit *words* named x, y, z. For example, we denote the three words of A by A_x, A_y, A_z. We will also use the expression *lane* to denote the concatenation of the four words which occupy the same position in each column, for example $(A_x|B_x|C_x|D_x)$ is the x lane.

The Gimli permutation consists of 24 rounds which are numbered starting at 24 and down to 1. Each round consists of between one and three steps:

- In all rounds, a non-linear 96-bit permutation or SP-box is applied on each of the columns.
- Every four rounds, starting at round 24, a *small swap* is applied, which consists of swapping the position of the words A_x and B_x and the words C_x and D_x. Additionally, a round constant is added to the word A_x. The round constant is $\mathrm{rc}_i = \texttt{9E377900} \oplus i$, where i is the round number.
- Every four rounds, starting at round 22, a *big swap* is applied, which consists of swapping the position of the words A_x and C_x and the words B_x and D_x.

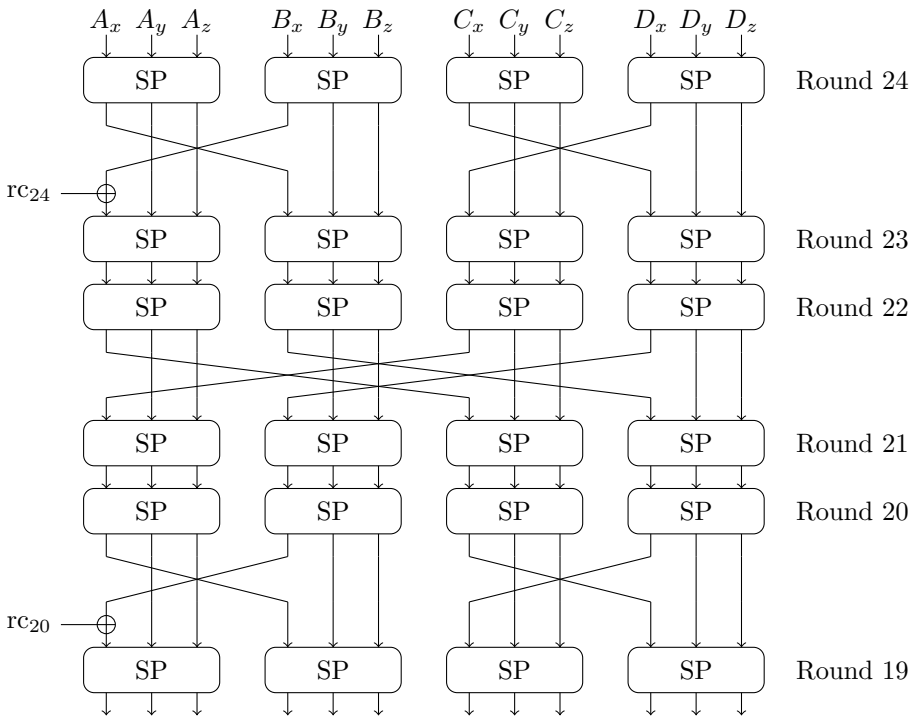


Figure 7.1: The first six rounds of the Gimli permutation.

The Gimli permutation is given as an algorithm in Algorithm 20, and the first six rounds are represented in Figure 7.1. The SP-box operates as follows on an input x, y, z:

1. Rotate x and y: $x \leftarrow x \lll 24,\ y \leftarrow y \lll 9$.
2. Perform the following non-linear operations in parallel:

   $$x \leftarrow x \oplus (z \ll 1) \oplus ((y \wedge z) \ll 2),$$
   $$y \leftarrow y \oplus x \oplus ((x \vee z) \ll 1),$$
   $$z \leftarrow z \oplus y \oplus ((x \wedge y) \ll 3).$$

   Note that shifts are used instead of rotations.
3. Swap x and z: $(x, z) \leftarrow (z, x)$.

Alternatively, here is the output $(x', y', z') = \mathrm{SP}(x, y, z)$ expressed using Boolean operators:

$$\begin{aligned}
x'_0 &= y_{23} + z_0, \\
x'_1 &= y_{24} + z_1, \\
x'_2 &= y_{25} + z_2, \\
x'_i &= y_{i-9} + z_i + x_{i+5}y_{i-12}, && 3 \le i \le 32, \\
y'_0 &= x_8 + y_{23}, \\
y'_i &= x_{i+8} + y_{i-9} + x_{i+7} + z_{i-1} + x_{i+7}z_{i-1}, && 1 \le i \le 32, \\
z'_0 &= x_8, \\
z'_1 &= x_9 + z_0, \\
z'_i &= x_{i+8} + z_{i-1} + y_{i-11}z_{i-2}, && 2 \le i \le 32.
\end{aligned}$$

7.2.2 Building a MILP model of the SP-box LAT

Let us now focus on the process of building a MILP model of the SP-box which describes its linear approximations. We will perform a slight abuse of notation and denote the 96 input mask variables by $(x_{31}, \ldots, x_0)$ for the x word, $(y_{31}, \ldots, y_0)$ for the y word, and $(z_{31}, \ldots, z_0)$ for the z word. For the output of the SP-box, we will denote the 96 variables associated to the mask by x'_i, y'_i, z'_i. We want to impose some constraints on these variables so that only valid linear approximations can be obtained. Since the

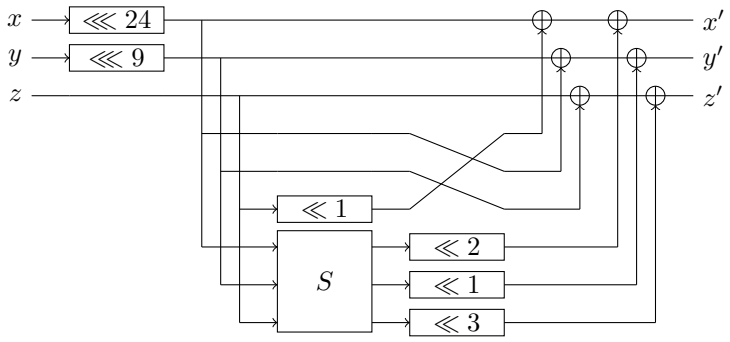


Figure 7.2: A circuit representation of the SP-box of Gimli where the non-linear part is contained inside an Sbox layer using a non-bijective 3-bit Sbox.

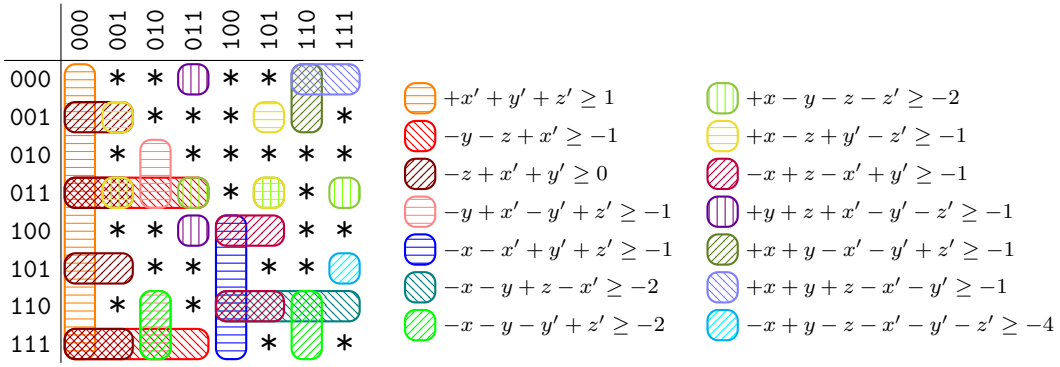


Figure 7.3: Karnaugh map used to model the LAT of the 3-bit Sbox S.

SP-box is a 96-bit permutation, it would be impractical to treat it like an Sbox and model its LAT directly.

Figure 7.2 depicts the application of the SP-box as a circuit. This representation has the advantage of containing all the nonlinear operations inside an Sbox layer which uses the three bit (non-bijective) Sbox S defined as $S(x, y, z) = (y \wedge z, x \vee z, x \wedge y)$. We will use this diagram as a basis for our MILP representation of the LAT of the SP-box.

The initial bit rotations can be easily represented in our model by shuffling the input activity variables accordingly. After the bit rotations, we reach a point in which the state is duplicated twice (so that three copies of the state remain). If we used the circuit approach of Subsubsection 7.1.1.1 to model these forking points, we would at first require $96 \cdot 3 = 288$ new bit activity variables and $4 \cdot 2 \cdot 96 = 768$ constraints.

However, we note that one of the three copies of the state will just have other material xored to it before becoming the output of the SP-box. Since the linear mask at all the branches of a XOR operation must be identical, we can use the output variables x'_i, y'_i, z'_i as one of the three copies of the state, thus saving 96 variables. Furthermore, another of the copies of the state is simply shuffled around and xored back into the state. This means we can also make the activity variables associated to this copy of the state equal to the (shuffled) activity variables for the output, thus saving another 96 variables. In fact, since one bit of this copy of the state is discarded, we also save four inequalities in the fork. In practice, these equalities can be either introduced directly into the model by substitution or simply left as equalities of two variables, as most presolvers will substitute them automatically.

We next have to model the Sbox layer. The 3-bit Sbox S can be represented using 14 inequalities found using a Karnaugh diagram, which is illustrated in Figure 7.3. All the non-zero entries of the LAT of S are ± 2, so we only require one activity indicator variable per Sbox. In total, we would require 96 additional bit activity variables for each of the Sbox outputs and 32 Sbox activity indicator variables. However, of the 96 outputs of the Sbox layer, 90 are xored to the SP-box output, and 6 are discarded. This means we can save all the bit activity variables and one Sbox activity indicator, as there is one Sbox which has all of its outputs discarded. Because of this Sbox, we can also remove three indicator activity variables from the fork at the beginning and their associated constraints. In terms of constraints, each of these 31 Sboxes requires $6 + 14$ inequalities, or $20 \cdot 31 = 620$ inequalities in total.

In the end, the LAT of the SP-box can be simulated with, in addition to the 192 input and output bit activity variables, 93 internal state bit activity variables and 31 Sbox activity indicator variables. The

model employs 1376 linear constraints.

7.2.3 Linear trails of the double SP-box

Before delving into the search for linear trails of the full Gimli permutation, we will briefly analyse the linear trails of the SP-box. Since the Gimli permutation mainly uses the composition of the SP-Box with itself (except for the first and the last rounds), an analysis of the linear properties of the "double" SP-Box SP^2 may provide some useful insight.

Let us consider that we apply the double SP-box to $A = (x, y, z)$ to obtain $A'' = (x'', y'', z'') = SP^2(x, y, z)$. By computing the components of A'' as Boolean functions of the components of A, it becomes readily apparent that the relationship $x_8 + x''_0 + y''_0 + z''_0 = 0$ always holds. This is a linear trail of the double SP-box with correlation 1, and it is unique.

We are also interested in other trails of high correlation different from ± 1. To this end, we used the MILP model of the SP-box. We found 41 different trails with correlation $\pm 2^{-1}$ and 572 trails with correlation $\pm 2^{-2}$, however, we should note that we did not count them exhaustively and thus more trails may exist. These high-correlation trails probably conform a very small fraction of all the possible linear approximations of the double SP-box (which is, after all, a 96-bit permutation). The problem of describing the distribution of the correlations of the linear trails of the double Gimli SP-box remains open. The linear trails of the double SP-box we identified in this way are not used in the linear trails we found for the Gimli permutation.

The linear trails which were found in the MILP search all had different input and output masks. For a couple of them, we also tried to find other trails with the same input and output masks, however we didn't find any with high correlation. This would suggest that, at least for these specific approximations of the double SP-box, there are no other significant trails and there should be no appreciable linear hull effect.

7.2.4 Linear trails of Gimli

We will now use the MILP model of the LAT of the Gimli SP-box to construct linear trails of reduced-round Gimli and eventually of the full permutation. Although Gimli is a permutation and thus features no key or key schedule, we will assume that both the piling-up lemma and the linear hull theorem hold. This means we can only obtain an estimate of the correlation of the linear approximations which have more than one dominant trail. A more accurate estimate of the correlation can be obtained by carefully considering the signs of the correlation contributions of each trail, but we were unable to do so in a reasonable amount of time due to some technical issues with running SCIP on our machine.

We will also consider that long, multiple-round linear trails of Gimli are likely to feature a single active SP-box in the central round. For this reason, we will initially focus on trails which have a single active SP-box in each round, and we will later try to extend them at the top and the bottom, this time allowing for more active SP-boxes. This restriction greatly limits the search space. Specifically, we will focus initially on linear trails for the iterated SP-box for which the mask for the x word is zero every two rounds. This means that the mask is unaffected by the big and small swaps, and these trails easily translate into trails for the reduced-round Gimli construction with the same correlation.

The MILP model for this iterated SP-box is simple to construct given the model for the SP-box itself, as it suffices to keep adding variables and constraints for each application of the SP-box. The condition that the x word has mask zero can be added quite quickly and easily by including an equality for each one of the associated activity variables, and letting the presolver substitute them in the model. If we wish to find linear trails which are also *iterative*, that is, which have the same input and output mask so that they can be concatenated with themselves, it suffices to include an equality for each pair of input and output activity variables, and again, letting the presolver handle the substitution.

We first look at iterative linear trails of the SP-box for which the x word is zero every two rounds. We searched for such trails for two, four and eight rounds. The found trails are represented in Figure 7.4. In the case of two rounds, the solver was able to prove that the optimal correlation is 2^{-26}, although we were unable to prove that the trail we found is the only one with that correlation. In the four-round case, the solver was unable to prove optimality after running for several days, but a trail with correlation 2^{-47} was found after a few hours. This trail provides better results than the two-round one, which has correlation 2^{-52} over four rounds.

We have also been able to construct an eight-round iterative trail with correlation 2^{-67}, which significantly improves on the four-round one, which has correlation 2^{-94} over eight rounds. Like with four rounds, we were unable to prove the optimality of this trail. However, we also performed some additional

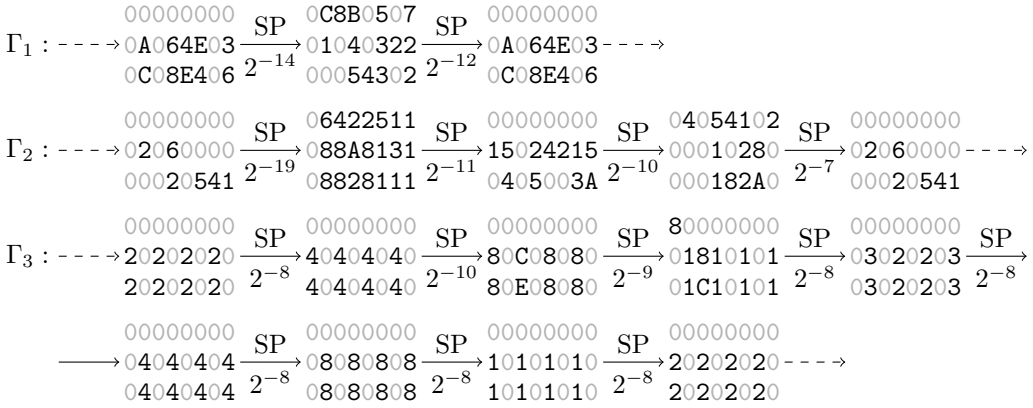


Figure 7.4: Iterative linear trails of the Gimli SP-box for two, four and eight rounds.

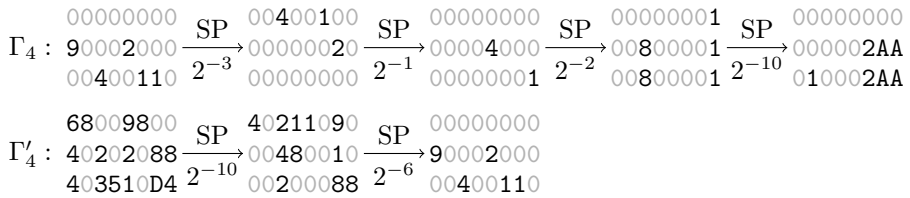


Figure 7.5: A four-round linear trail of the SP-box and a two-round extension.

experiments searching for additional eight-round trails which have the same input and output masks. We found that, in fact, there are 8 trails with correlation 2^{-67}, 32 trails with correlation 2^{-68}, 128 trails with correlation 2^{-69}, 466 trails with correlation 2^{-70}, and 1527 trails with correlation 2^{-71}. If we considered a key-alternating block cipher version of Gimli this would give the linear approximation a linear potential of at least $2^{-128.8}$ for the eight-round iterative linear approximation, in comparison to the value of 2^{-134} which would correspond to a single linear trail.

For a small number of rounds, however, there are trails which showcase considerably higher correlations than the iterative ones. For this reason, we provide higher correlation, non-iterative trails for up to six rounds in Figure 7.5. We first searched for four-round trails of the SP-box with a zero mask in the x word every two rounds, and found that the optimal correlation is 2^{-16}. We then tried to extend this trail by an additional two rounds at the top. This was done by taking a two-round model and fixing the values of the activity indicator variables at the output mask. There are no double SP-box extensions for which the input mask has the x word set to zero. However, after removing that condition we found a two-round extension with correlation 2^{-16}, thus providing a six round linear trail with correlation 2^{-32}.

After constructing a few short trails, we will now explain how we found a linear trail for the full 24-round Gimli permutation. We start from a 16-round trail which is the concatenation of the eight-round iterative trail with itself, and has correlation 2^{-134}. From the analysis of other compatible linear trails to the eight-round iterative approximation, we know that this 16-round approximation has an ELP of at least $2^{-257.6}$.

Starting from this 16-round linear approximation, we search for extensions which reach the full 24 rounds. We can extend the approximation at the end with one additional round at a correlation of 2^{-8} (this round has the same transition as the first round of Γ_3). The purpose of this one-round extension at the end is to line up the 24-round approximation with the construction of the Gimli permutation, which starts with a single SP-box layer and continues with a small swap. At the top of the approximation, we search for an optimal 7-round extension which will cover rounds 24 to 18. In order to make the search space smaller, we impose the condition that the x word at the input mask to round 19 is zero, although we still consider the small swap at round 24 and the big swap at round 22. Under these specific constraints, we find the optimal extension has correlation 2^{-47}.

Together, these extensions conform a 24-round linear trail with correlation 2^{-189}. Furthermore, because of the linear hull properties of the iterative linear trail, we know that the ELP of a linear approximation using the same input and output linear masks for a key-alternating block cipher version of Gimli would be at least $2^{-367.6}$.

Our best linear trails for up to 24 rounds of Gimli are condensed in Table 7.1. In most cases, the optimality of these trails has not been proven (and for larger number of rounds, it is very unlikely). This

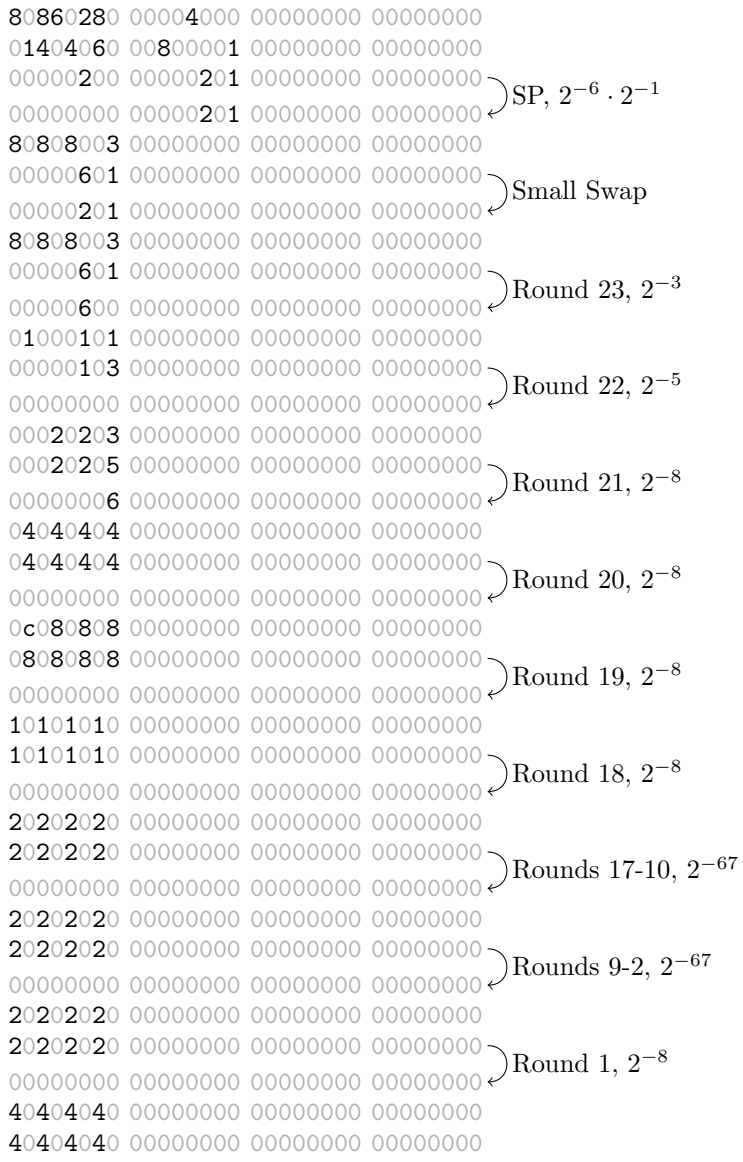


Figure 7.6: The full 24-round Gimli linear trail Γ_5 with correlation 2^{-189}.

Table 7.1: Linear trails for Gimli. Some of them apply to shifted versions of the algorithm starting with two consecutive SP-box substitutions instead of one.

# Rounds	Correlation	Construction	Shift
1	1	Probability 1 trail	No
2	1	Probability 1 trail	Yes
4	2^{-12}	Round 1 of Γ'_4, rounds 4 to 2 of Γ_4	No
6	2^{-32}	Γ'_4 and Γ_4	Yes
8	2^{-55}	Rounds 24 to 17 of Γ_5	No
12	2^{-90}	Rounds 24 to 13 of Γ_5	No
16	2^{-122}	Rounds 24 to 9 of Γ_5	No
20	2^{-157}	Rounds 24 to 5 of Γ_5	No
24	2^{-189}	Γ_5	No

is due to both the fact that our search has been limited to specific trail families, such as the ones which stay contained within a single column, as well as the fact that in most cases we didn't wait until the MILP solver proved that the current solution is actually optimal.

This 24-round linear approximation can be used to mount a distinguishing attack on the Gimli permutation with a data complexity of $\mathcal{O}\left(2^{378}\right)$. In fact, because of the presence of many trails in the linear hull of Γ_3, it is quite likely that $\mathcal{O}\left(2^{367.6}\right)$ known plaintext-ciphertext pairs will suffice. It is possible to reduce the complexity slightly by using multiple linear cryptanalysis, since there are four versions of the approximation we can use (one for each column).

Each of the optimization problems mentioned in this section was solved on a laptop equipped with a U-series Intel processor (a consumer CPU mostly oriented towards power efficiency). Problems which had undetermined input and output masks were solved with a running time of around 20 minutes for the lightest ones and half a day for the most complex ones. Optimization problems dealing with extending a given output mask "backwards" usually ran a lot faster, sometimes being solved in less than one minute. Optimization problems dealing with extending a given input mask "forwards" seemed a lot more difficult, and some took up to four hours to solve. As of now I am unaware of the reasons for this notable difference in performance.

7.3 Conclusion and Open Problems

All the cryptanalysis of the Gimli permutation we presented in [FLN+20, FLN+21] has no direct impact on the security of the primitives which are built on it. However, the Gimli proposal was not selected as a Lightweight Cryptography finalist by the NIST. Regarding linear cryptanalysis, the problem of finding a linear approximation with input and output masks on the x lane (which is the rate of the Gimli-hash and Gimli-cipher constructions) remains open.

Conclusion

In recent years, the field of symmetric cryptology has seen a shift towards an increased interest in lightweight cryptography applications. This change has been motivated by the requirements of the electronics and computer industries: our lives are becoming populated by more and more connected devices, some of which have very limited processing capabilities, but which must still be able to communicate securely with other devices. Although cryptography has always cared about efficiency, these application scenarios require a major shift in perspective for the designers: ideally, we want to achieve the desired level of security without wasting a single instruction or transistor. The importance of this emerging field is exemplified by the NIST's Lightweight Cryptography project.

In the design environment of block ciphers oriented towards lightweight cryptography, the correct evaluation of security margins becomes paramount. As we start to prioritise cost, we naturally shift towards slimmer security margins in order to reduce the number of rounds. For this reason, it becomes extremely important to accurately evaluate how many rounds of the cipher can be covered by a key recovery attack. It is particularly in this context that the importance of understanding the best possible key recovery attack algorithms becomes clear. Furthermore, it seems that lightweight primitives might be more likely to present key recovery attacks over multiple rounds due to the slower diffusion.

On the other hand, these improved algorithms for key recovery attacks seem to require a lot of input by the cryptanalyst and result in fairly technical and complicated attacks, and it seems unreasonable for designers to spend so much time coming up with the best possible attacks on their constructions. This highlights the need to create automatic tools which are able to apply these advanced techniques with little to no external intervention. This development of automatic, easy-to-use tools has already started in the case of distinguishers, with techniques based on MILP, SAT and CP solvers. There is also a need for literature on these topics which is practically-oriented, and which can act as a guide for designers.

Finally, there are some open problems which are related to comparing different techniques: for example, in the case of using MILP modelling to search for differential characteristics or linear trails, there are multiple ways of modelling Sboxes which are described in the literature. As a designer who wants to implement a model of their design, it can be very unclear which of these techniques will produce the best results. This suggests the need for a thorough experimental comparison of the different methods.

Regarding my own future research, I would to continue working on the development and refinement of these techniques, but I would also like to improve our understanding of existing techniques, and to make them as easy-to-use as possible.

Bibliography

[AÅBL12] Mohamed Ahmed Abdelraheem, Martin Ågren, Peter Beelen, and Gregor Leander. On the distribution of linear biases: Three instructive examples. In *Advances in Cryptology - CRYPTO 2012, Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2012.

[Abd12] Mohamed Ahmed Abdelraheem. Estimating the probabilities of low-weight differential and linear approximations on present-like ciphers. In *Information Security and Cryptology - ICISC 2012, Revised Selected Papers*, volume 7839 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 2012.

[AES01] *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. U.S. Department of Commerce, National Institute of Standards and Technology, 2001.

[AIK+00] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In *Selected Areas in Cryptography, SAC 2000, Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[AKM97] Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. Best differential characteristic search of FEAL. In *Fast Software Encryption, FSE '97, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 1997.

[Ama07] Kozue Amano. Navigation 48: Traghetto. In *ARIA*, volume 10. Mag Garden, 2007.

[AOS00] Rogerio G. Alves, P.L. Osorio, and Meghana N.S. Swamy. General FFT pruning algorithm. In *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems (Cat.No.CH37144)*, volume 3, pages 1192–1195 vol.3, 2000.

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Transactions on Symmetric Cryptology*, 2017(4):99–129, 2017.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In *Fast Software Encryption, FSE '98, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[BBC+22] Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. Improved differential-linear attacks with applications to arx ciphers. *Journal of Cryptology (accepted)*, 2022.

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. *Journal of Cryptology*, 18(4):291–311, 2005.

[BC20] Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Transactions on Symmetric Cryptology*, 2020(3):327–361, 2020.

[BCD+22] Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. New attacks from old distinguishers improved attacks on serpent. In *Topics in Cryptology - CT-RSA 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 484–510. Springer, 2022.

[BCF+21] Marek Broll, Federico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, and María Naya-Plasencia. Generic framework for key-guessing improvements. In *Advances in Cryptology - ASIACRYPT 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 453–483. Springer, 2021.

[BCQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In *Advances in Cryptology - CRYPTO 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2004.

[BDD+15] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban. Cryptanalysis of SP networks with partial non-linear layers. In *Advances in Cryptology - EUROCRYPT 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 315–342. Springer, 2015.

[BDK01a] Eli Biham, Orr Dunkelman, and Nathan Keller. Linear cryptanalysis of reduced round serpent. In *Fast Software Encryption, FSE 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2001.

[BDK01b] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the serpent. In *Advances in Cryptology - EUROCRYPT 2001, Proceedings*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[BDK02a] Eli Biham, Orr Dunkelman, and Nathan Keller. Enhancing differential-linear cryptanalysis. In *Advances in Cryptology - ASIACRYPT 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2002.

[BDK02b] Eli Biham, Orr Dunkelman, and Nathan Keller. New results on boomerang and rectangle attacks. In *Fast Software Encryption, FSE 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[BDK03] Eli Biham, Orr Dunkelman, and Nathan Keller. Differential-linear cryptanalysis of serpent. In *Fast Software Encryption, 10th International Workshop, FSE 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 9–21. Springer, 2003.

[BDK05a] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-key boomerang and rectangle attacks. In *Advances in Cryptology - EUROCRYPT 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 507–525. Springer, 2005.

[BDK05b] Eli Biham, Orr Dunkelman, and Nathan Keller. A related-key rectangle attack on the full KASUMI. In *Advances in Cryptology - ASIACRYPT 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2005.

[BDKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A new tool for differential-linear cryptanalysis. In *Advances in Cryptology - EUROCRYPT 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 313–342. Springer, 2019.

[BDPA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. In *ECRYPT Hash Workshop*, 2007.

[Ben12] R.L. Benson. *The Venona Story*. CreateSpace Independent Publishing Platform, 2012.

[Ber08] Daniel J. Bernstein. The salsa20 family of stream ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[BHV14] Aslí Bay, Jialin Huang, and Serge Vaudenay. Improved linear cryptanalysis of reduced-round MIBS. In *Advances in Information and Computer Security - IWSEC 2014, Proceedings*, volume 8639 of *Lecture Notes in Computer Science*, pages 204–220. Springer, 2014.

[Bih94] Eli Biham. On matsui's linear cryptanalysis. In *Advances in Cryptology - EUROCRYPT '94, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 1994.

[BJV04] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In *Advances in Cryptology - ASIACRYPT 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2004.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BKL+11] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. spongent: A lightweight hash function. In *Cryptographic Hardware and Embedded Systems - CHES 2011, Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, 2011.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In *Cryptographic Hardware and Embedded Systems - CHES 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 299–320. Springer, 2017.

[BKL+19] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli. Submission to the NIST Lightweight Cryptography project. `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/gimli-spec.pdf`., 2019.

[BLT20] Christof Beierle, Gregor Leander, and Yosuke Todo. Improved differential-linear attacks with applications to ARX ciphers. In *Advances in Cryptology - CRYPTO 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[BN13] Céline Blondeau and Kaisa Nyberg. New links between differential and linear cryptanalysis. In *Advances in Cryptology - EUROCRYPT 2013, Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 388–404. Springer, 2013.

[BN15] Céline Blondeau and Kaisa Nyberg. Joint data and key distribution of the linear cryptanalysis test statistic and its impact to data complexity estimates of multiple/multidimensional linear and truncated differential attacks. *IACR Cryptology ePrint Archive*, page 935, 2015.

[BN16] Céline Blondeau and Kaisa Nyberg. Improved parameter estimates for correlation and capacity deviates in linear cryptanalysis. *IACR Transactions on Symmetric Cryptology*, 2016(2):162–191, 2016.

[BN17] Céline Blondeau and Kaisa Nyberg. Joint data and key distribution of simple, multiple, and multidimensional linear cryptanalysis test statistic and its impact to data complexity. *Designs, Codes and Cryptography*, 82(1-2):319–349, 2017.

[BP18] Eli Biham and Stav Perle. Conditional linear cryptanalysis - cryptanalysis of DES with less than 242 complexity. *IACR Transactions on Symmetric Cryptology*, 2018(3):215–264, 2018.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In *Advances in Cryptology - CRYPTO '90, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of FEAL and N-Hash. In *Advances in Cryptology - EUROCRYPT '91, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1991.

[BS92] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology - CRYPTO '92, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, page 404, 2013.

[BT13] Andrey Bogdanov and Elmar Tischhauser. On the wrong key randomisation and key equivalence hypotheses in matsui's algorithm 2. volume 8424 of *Lecture Notes in Computer Science*, pages 19–38. Springer, 2013.

[BTV18] Andrey Bogdanov, Elmar Tischhauser, and Philip S. Vejre. Multivariate profiling of hulls for linear cryptanalysis. *IACR Transactions Symmetric on Cryptology*, 2018(1):101–125, 2018.

[BV17] Andrey Bogdanov and Philip S. Vejre. Linear cryptanalysis of DES with asymmetries. In *Advances in Cryptology - ASIACRYPT 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 187–216. Springer, 2017.

[BW99] Alex Biryukov and David A. Wagner. Slide attacks. In *Fast Software Encryption FSE '99, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[BW12] Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 29–48. Springer, 2012.

[Car21] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021.

[CHN08] Joo Yeon Cho, Miia Hermelin, and Kaisa Nyberg. A new technique for multidimensional linear cryptanalysis with applications on reduced round serpent. In *Information Security and Cryptology - ICISC 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 2008.

[Cho10] Joo Yeon Cho. Linear cryptanalysis of reduced-round PRESENT. In *Topics in Cryptology - CT-RSA 2010, Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In *Advances in Cryptology - EUROCRYPT 2018, Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714. Springer, 2018.

[CHW08] Huiju Cheng, Howard M. Heys, and Cheng Wang. PUFFIN: A novel compact block cipher targeted to embedded digital systems. In *11th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2008, Proceedings*, pages 383–390. IEEE Computer Society, 2008.

[CI17] Mahdi Cheraghchi and Piotr Indyk. Nearly optimal deterministic algorithm for sparse walsh-hadamard transform. *ACM Transactions on Algorithms*, 13(3):34:1–34:36, 2017.

[CMTY15] David Castro-Palazuelos, Modesto Medina-Melendrez, Deni Torres-Roman, and Shkvarko Yuriy. Unified commutation-pruning technique for efficient computation of composite DFTs. *EURASIP Journal on Advances in Signal Processing*, 11-2015, 11 2015.

[CNV13] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In *Advances in Cryptology - CRYPTO 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2013.

[CS09] Baudoin Collard and François-Xavier Standaert. A statistical saturation attack against the block cipher PRESENT. In *Topics in Cryptology - CT-RSA 2009, Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2009.

[CSQ07a] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improved and multiple linear cryptanalysis of reduced round serpent. In *Information Security and Cryptology, Inscrypt 2007, Revised Selected Papers*, volume 4990 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2007.

[CSQ07b] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the time complexity of matsui's linear cryptanalysis. In *Information Security and Cryptology - ICISC 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2007.

[CSQ08] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Experiments on the multiple linear cryptanalysis of reduced round serpent. In *Fast Software Encryption, FSE 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 382–397. Springer, 2008.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[CWZ+19] Jiahao Cai, Zihao Wei, Yingjie Zhang, Siwei Sun, and Lei Hu. Zero-sum distinguishers for round-reduced GIMLI permutation. In *5th International Conference on Information Systems Security and Privacy, ICISSP 2019, Proceedings*, pages 38–43. SciTePress, 2019.

[DES77] *Data Encryption Standard (DES)*. Federal Information Processing Standards Publication 46-3. U.S. Department of Commerce, National Institute of Standards and Technology, reaffirmed 1988,1993,1999, withdrawn 2005, 1977.

[DGM+19] Nilanjan Datta, Ashrujit Ghoshal, Debdeep Mukhopadhyay, Sikhar Patranabis, Stjepan Picek, and Rajat Sadhukhan. Trifle, 2019.

[DGV94] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In *Fast Software Encryption: Second International Workshop, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 275–285. Springer, 1994.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DH77] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.

[DIK08] Orr Dunkelman, Sebastiaan Indesteege, and Nathan Keller. A differential-linear attack on 12-round serpent. In *Progress in Cryptology - INDOCRYPT 2008, Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 308–321. Springer, 2008.

[DM90] Dan E. Dudgeon and Russell M. Mersereau. *Multidimensional Digital Signal Processing*. Prentice Hall Professional Technical Reference, 1990.

[DPVAR00] Joan Daemen, Michael Peeters, Gilles Van Assche, and Vincent Rijmen. *The NOEKEON block cipher*. Proposal to the NESSIE Project, 2000.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer Berlin Heidelberg, 2002.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *Journal of Mathematical Cryptology*, 1(3):221–242, 2007.

[EM91] Shimon Even and Yishay Mansour. A construction of a cioher from a single pseudorandom permutation. In *Advances in Cryptology - ASIACRYPT '91, Proceedings*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1991.

[EMST76] William F. Ehrsam, Carl H. W. Meyer, John L. Smith, and Walter L. Tuchman. Message verification and transmission error detection by block chaining, U.S. Patent 4074066A, April 1976.

[ER08] Jonathan Etrog and Matthew J. B. Robshaw. The cryptanalysis of reduced-round SMS4. In *Selected Areas in Cryptography, SAC 2008, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2008.

[Eve87] Jan-Hendrik Evertse. Linear structures in blockciphers. In *Advances in Cryptology - EUROCRYPT '87, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 1987.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[FLN+20] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sibleyras. New results on Gimli: Full-permutation distinguishers and improved collisions. In *Advances in Cryptology - ASIACRYPT 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2020.

[FLN+21] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sibleyras. Internal symmetries and linear properties: Full-permutation distinguishers and improved collisions on Gimli. *Journal of Cryptology*, 34(4):45, 2021.

[Fló22] Antonio Flórez-Gutiérrez. Optimising linear key recovery attacks with affine walsh transform pruning. In *Advances in Cryptology - ASIACRYPT 2022*, 2022.

[FN20] Antonio Flórez-Gutiérrez and María Naya-Plasencia. Improving key-recovery in linear attacks: Application to 28-round PRESENT. In *Advances in Cryptology - EUROCRYPT 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 221–249. Springer, 2020.

[FS03] Niels Ferguson and Bruce Schneier. *Practical cryptography*. Wiley, 2003.

[GOS89] *Federal Information Processing Standard - Cryptographic Algorithm - GOST*. 28147-89. Russian National Bureau of Standards, 1989.

[GS66] W. Morven Gentleman and G. Sande. Fast fourier transforms: For fun and profit. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS '66 (Fall), page 563–578, New York, NY, USA, 1966. Association for Computing Machinery.

[GT78] E. K. Grossman and B. Tuckerman. Analysis of a weakened feistel-like cipher. In *1978 International Conference on Communications*, pages 46.3.1 – 46.3.5. Alger Press Limited, 1978.

[Ham17] Mike Hamburg. Cryptanalysis of 22 1/2 rounds of Gimli. *IACR Cryptology ePrint Archive*, page 743, 2017.

[HCN08] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis of reduced round serpent. In *Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Proceedings*, volume 5107 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2008.

[HCN19] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis. *Journal of Cryptology*, 32(1):1–34, 2019.

[HSK02] Yasuo Hatano, Hiroki Sekine, and Toshinobu Kaneko. Higher order differential attack of camellia (II). In *Selected Areas in Cryptography, SAC 2002, Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 2002.

[HV18] Mathias Hall-Andersen and Philip S. Vejre. Generating graphs packed with paths estimation of linear approximations and differentials. *IACR Transactions on Symmetric Cryptology*, 2018(3):265–289, 2018.

[HW05] Zhong Hu and Honghui Wan. A novel generic fast Fourier transform pruning technique and complexity analysis. *IEEE Transactions on Signal Processing*, 53(1):274–282, 2005.

[JSD01] Dragan Jankovic, Radomir S. Stankovic, and Rolf Drechsler. Decision diagram method for calculation of pruned walsh transform. *IEEE Transactions on Computers*, 50(2):147–157, 2001.

[JSZW09] Jorge Nakahara Jr., Pouyan Sepehrdad, Bingsheng Zhang, and Meiqin Wang. Linear (hull) and algebraic cryptanalysis of the block cipher PRESENT. In *Cryptology and Network Security, CANS 2009, Proceedings*, volume 5888 of *Lecture Notes in Computer Science*, pages 58–75. Springer, 2009.

[Jun01] Pascal Junod. On the complexity of Matsui's attack. In *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2001.

[JZZD20] Fulei Ji, Wentao Zhang, Chunning Zhou, and Tianyou Ding. Improved (related-key) differential cryptanalysis on GIFT. *IACR Cryptology ePrint Archive*, page 1242, 2020.

[Kah96] David Kahn. *The Codebreakers: The Story of Secret Writing.* Scribner, 1967,1996.

[Kar53] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

[KB96] Lars R. Knudsen and Thomas A. Berson. Truncated differentials of SAFER. In *Fast Software Encryption 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 15–26. Springer, 1996.

[Ker83] Auguste Kerchkoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9:5–38, 1983.

[KKHS08] Tae Hyun Kim, Jongsung Kim, Seokhie Hong, and Jaechul Sung. Linear and differential cryptanalysis of reduced SMS4 block cipher. *IACR Cryptology ePrint Archive*, page 281, 2008.

[KKS00] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and serpent. In *Fast Software Encryption, 7th International Workshop, FSE 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

[KM01] Masayuki Kanda and Tsutomu Matsumoto. Security of camellia against truncated differential cryptanalysis. In *Fast Software Encryption, FSE 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2001.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In *Fast Software Encryption: Second International Workshop, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Knu98] Lars Knudsen. Deal - a 128-bit block cipher. In *NIST AES Proposal*, 1998.

[KR94] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear cryptanalysis using multiple approximations. In *Advances in Cryptology - CRYPTO '94, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. Springer, 1994.

[Lan93] Serge Lang. *Algebra.* Addison-Wesley (3rd Edition Springer, 2002), 1993.

[LC14] Mingjie Liu and Jiazhe Chen. Improved linear attacks on the chinese block cipher standard. *Journal of Computer Science Technology*, 29(6):1123–1133, 2014.

[Lea11] Gregor Leander. On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In *Advances in Cryptology - EUROCRYPT 2011, Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 303–322. Springer, 2011.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In *Advances in Cryptology - CRYPTO '94, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 1994.

[LHL+01] Seonhee Lee, Seokhie Hong, Sangjin Lee, Jongin Lim, and Seonhee Yoon. Truncated differential cryptanalysis of camellia. In *Information Security and Cryptology - ICISC 2001, Proceedings*, volume 2288 of *Lecture Notes in Computer Science*, pages 32–38. Springer, 2001.

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. Exploiting weak diffusion of Gimli: Improved distinguishers and preimage attacks. *IACR Transactions on Symmetric Cryptology*, 2021(1):185–216, 2021.

[LLL21] Meicheng Liu, Xiaojuan Lu, and Dongdai Lin. Differential-linear cryptanalysis from an algebraic perspective. In *Advances in Cryptology - CRYPTO 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 247–277. Springer, 2021.

[LM90] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology - EUROCRYPT '90, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology - EUROCRYPT '91, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[Mar71] J. Markel. FFT pruning. *IEEE Transactions on Audio and Electroacoustics*, 1971.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[Mat94a] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology - CRYPTO '94, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1994.

[Mat94b] Mitsuru Matsui. On correlation between the order of S-boxes and the strength of DES. In *Advances in Cryptology - EUROCRYPT '94, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[MC13] James McLaughlin and John A. Clark. Filtered nonlinear cryptanalysis of reduced-round serpent, and the wrong-key randomization hypothesis. In *Cryptography and Coding - IMACC 2013, Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 120–140. Springer, 2013.

[MHL+02] Dukjae Moon, Kyungdeok Hwang, Wonil Lee, Sangjin Lee, and Jongin Lim. Impossible differential cryptanalysis of reduced round XTEA and TEA. In *Fast Software Encryption, FSE 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2002.

[Mil82] Frank Miller. *Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams*. C.M. Cornwell, 1882.

[MJ56] E. J. McCluskey Jr. Minimization of boolean functions*. *Bell System Technical Journal*, 35(6):1417–1444, 1956.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - Inscrypt 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[MY92] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In *Advances in Cryptology - EUROCRYPT '92, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1992.

[Nag86] Keinosuke Nagai. Pruning the decimation-in-time FFT algorithm with frequency shift. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):1008–1010, 1986.

[NF22] Kaisa Nyberg and Antonio Flórez-Gutiérrez. Linear cryptanalysis. In *Encyclopedia of Cryptology*. 2022.

[NIS01] *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. Federal Information Processing Standards Special Publication 800-38A. U.S. Department of Commerce, National Institute of Standards and Technology, 2001.

[NP33] J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231:289–337, 1933.

[NWW11] Phuong Ha Nguyen, Hongjun Wu, and Huaxiong Wang. Improving the algorithm 2 in multidimensional linear cryptanalysis. In *Information Security and Privacy - ACISP 2011, Proceedings*, volume 6812 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2011.

[Nyb94] Kaisa Nyberg. Linear approximation of block ciphers. In *Advances in Cryptology - EUROCRYPT '94, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 439–444. Springer, 1994.

[Nyb17] Kaisa Nyberg. Statistical and linear independence of binary random variables. *IACR Cryptology ePrint Archive*, page 432, 2017.

[Ohk09] Kenji Ohkuma. Weak keys of reduced-round PRESENT for linear cryptanalysis. In *Selected Areas in Cryptography, SAC 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2009.

[PM92] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Pearson (5th Edition 2021), 1992.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SB93] Henrik V. Sorensen and C. Sydney Burrus. Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*, 41(3):1184–1200, 1993.

[Sch93] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, FSE 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[Sch96] Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition*. Wiley, 1996.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, 2008.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[SHA15] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standards Publication 202. U.S. Department of Commerce, National Institute of Standards and Technology, 2015.

[SHW+14a] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, and Ling Song. Automatic enumeration of (related-key) differential and linear characteristics with predefined properties and its applications. *IACR Cryptology ePrint Archive*, page 747, 2014.

[SHW+14b] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In *Advances in Cryptology - ASIACRYPT 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

[Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30(5):776–780, 1984.

[SK99] Haruki Seki and Toshinobu Kaneko. Cryptanalysis of five rounds of CRYPTON using impossible differentials. volume 1716 of *Lecture Notes in Computer Science*, pages 43–51. Springer, 1999.

[Ski76] David P. Skinner. Pruning the decimation in-time FFT algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(2):193–194, 1976.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. 01 1998.

[SM87] Akihiro Shimizu and Shoji Miyaguchi. Fast data encipherment algorithm FEAL. In *Advances in Cryptology - EUROCRYPT '87, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 1987.

[SR79] T.V. Sreenivas and P.V.S. Rao. FFT algorithm for both input and output pruning. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(3):291–292, 1979.

[SS05] S. Singh and S. Srinivasan. Architecturally efficient FFT pruning algorithm. *Electronics Letters*, 41(23):1–2, Nov 10 2005.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In *Fast Software Encryption, FSE 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[ST96] He Shousheng and Mats Torkelson. Computing partial DFT for comb spectrum evaluation. *IEEE Signal Processing Letters*, 3(6):173–175, 1996.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Transactions on Symmetric Cryptology*, 2021(1):269–315, 2021.

[Ter99] Audrey Terras. *Fourier Analysis on Finite Groups and Applications*. London Mathematical Society Student Texts. Cambridge University Press, 1999.

[TG91] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In *Advances in Cryptology - CRYPTO '91, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 172–181. Springer, 1991.

[TTS+08] Yukiyasu Tsunoo, Etsuko Tsujihara, Maki Shigeri, Teruo Saito, Tomoyasu Suzaki, and Hiroyasu Kubo. Impossible differential cryptanalysis of CLEFIA. In *Fast Software Encryption, FSE 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 398–411. Springer, 2008.

[Ver19] Gilbert S. Vernam. Secret signaling system, U.S. Patent 1310719A, July 1919.

[VG18] Stefan Vigerske and Ambros M. Gleixner. SCIP: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, 33(3):563–593, 2018.

[Wag99] David A. Wagner. The boomerang attack. In *Fast Software Encryption, 6th International Workshop, FSE '99, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[WZSL12] Linkai Wang, Xiaofang Zhou, Gerald E. Sobelman, and Ran Liu. Generic mixed-radix FFT pruning. *IEEE Signal Processing Letters*, 19(3):167–170, 2012.

[YH97] Rao K. Yarlagadda and John E. Hershey. *Hadamard Matrix Analysis and Synthesis - With Applications to Communications and Signal/Image Processing*, volume 383 of *The Springer International Series in Engineering and Computer Science*. Springer, 1997.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A bit-slice ultra-lightweight block cipher suitable for multiple platforms. *IACR Cryptology ePrint Archive*, page 84, 2014.

[ZZ15] Lei Zheng and Shao-Wu Zhang. Fft-based multidimensional linear attack on PRESENT using the 2-bit-fixed characteristic. *Security and Communication Networks*, 8(18):3535–3545, 2015.

This thesis proposes improvements which can be applied to several techniques for the cryptanalysis of symmetric primitives. Special attention is given to linear cryptanalysis, for which a technique based on the fast Walsh transform was already known (Collard et al., ICISC 2007). We introduce a generalised version of this attack, which allows us to apply it on key recovery attacks over multiple rounds, as well as to reduce the complexity of the problem using information extracted, for example, from the key schedule. We also propose a general technique for speeding key recovery attacks up which is based on the representation of Sboxes as binary decision trees. Finally, we showcase the construction of a linear approximation of the full version of the Gimli permutation using mixed-integer linear programming (MILP) optimisation.

Keywords: cryptanalysis, symmetric cryptography, linear cryptanalysis, Walsh transform, block cipher, PRESENT